

Prepared for the  
GEORGE C. MARSHALL  
SPACE FLIGHT CENTER  
Huntsville, Alabama

13 September 1974

Contract No. NAS8-30538  
MSFC No. MSFC DRL-427A, Line Item 003  
IBM No. 74W-00224

## Study on Spacelab Software Development and Integration Concepts

### Final Technical Report

(NASA-CR-120472) STUDY ON SPACELAB  
SOFTWARE DEVELOPMENT AND INTEGRATION  
CONCEPTS Final Technical Report

(International Business Machines Corp.)

302 p HC \$18.25

CSCD 22B G3/31

N74-34338

Unclas  
50715

IBM

Prepared for the  
GEORGE C. MARSHALL  
SPACE FLIGHT CENTER  
Huntsville, Alabama

13 September 1974

Contract No. NAS8-30538  
MSFC No. MSFC DRL-427A, Line Item 003  
IBM No. 74W-00224

## Study on Spacelab Software Development and Integration Concepts

### Final Technical Report

Classification and Content Approval

Robert W. Draney

Data Manager Approval

A. Radkowski

Program Office Approval

Lred O. Heddens



Federal Systems Division, Civil and Space Systems/Huntsville, Alabama

## **NOTICE**

**Questions or comments concerning the contents of this report should be directed to:**

**Software Systems  
Department 222  
IBM, Space Systems  
Federal Systems Division  
150 Sparkman Drive  
Huntsville, Alabama 35806**

**Phone: (205) 837-4000 Extension 2250**

## **ABSTRACT**

The Space Transportation System's capability of multiple flights and rapid ground turnaround has provided new challenges in payload planning and processing. Spacelab flexibility in payload accommodations, coupled with the very high projected Spacelab missions, provides a new dimension in payload ground operations and software development.

This five-month study addressed the critical areas of Experiment Flight Applications software development, Spacelab Test and Checkout concepts, software integration concepts, and software test and integration facility requirements. The intent of the study was to bound requirements and sizing of the total Spacelab software development efforts to give Marshall Space Flight Center the necessary information for Spacelab software development planning.

The study on Spacelab Software Development and Integration concepts was performed under the technical direction of Messrs. J. Turner and J. Christy, Data Systems Laboratory, Data Systems Development Division, and Software Development Branch.



# TABLE OF CONTENTS

SECTION		PAGE
1	STUDY SYNOPSIS. . . . .	1-1
1.1	STUDY INTRODUCTION . . . . .	1-1
1.2	MAJOR STUDY CONCLUSIONS . . . . .	1-5
1.3	STUDY RECOMMENDATIONS . . . . .	1-9
1.4	STUDY BASELINE . . . . .	1-11
2	TASK 2B: DEFINITION OF SPACELAB EXPERIMENT SOFTWARE DEVELOPMENT CONCEPTS . . . . .	2-1
2.1	TASK 2B: SUMMARY . . . . .	2-1
2.2	TRAFFIC MODEL & SCHEDULE ASSUMPTIONS ANALYSES . . . . .	2-5
2.3	PI TO PI FLOW ASSESSMENT . . . . .	2-13
2.4	PI CDMS SOFTWARE INTERFACES . . . . .	2-19
2.5	EXPERIMENT CHECKOUT/VERIFICATION TECHNIQUES . . . . .	2-22
2.6	DEVELOPMENT PHILOSOPHY DETERMINATION . . . . .	2-29
2.7	EXPERIMENT SOFTWARE REQUIREMENTS . . . . .	2-47
2.8	DEVELOPMENT TOOLS REQUIREMENTS . . . . .	2-53
2.9	STIL REQUIREMENTS . . . . .	2-63
3	TASK 3B: IDENTIFY THE SPACELAB TEST AND CHECKOUT SOFTWARE CONCEPTS . . . . .	3-1
3.1	TASK 3B: SUMMARY . . . . .	3-1
3.2	ANALYSIS OF SPACELAB TEST AND CHECKOUT . . . . .	3-5
3.3	TEST AND CHECKOUT SOFTWARE CONCEPTS . . . . .	3-13
3.4	TEST AND CHECKOUT SOFTWARE DESIGN CONSIDERATIONS . . . . .	3-27
3.5	TEST AND CHECKOUT SOFTWARE MAINTENANCE REQUIREMENTS . . . . .	3-31
3.6	STIL REQUIREMENTS . . . . .	3-37
4	TASK 4B: MISSION OPERATIONS . . . . .	4-1
4.1	TASK 4B: SUMMARY . . . . .	4-1
4.2	MISSION SUPPORT SYSTEM DESCRIPTION . . . . .	4-3
4.3	DATA FLOW ANALYSIS . . . . .	4-9
4.4	MISSION PLANNING SYSTEM (MPS) . . . . .	4-11
4.5	PI PARTICIPATION IN MISSION OPERATIONS CONCEPT . . . . .	4-17
4.6	STIL SUPPORT REQUIREMENTS . . . . .	4-21

# TABLE OF CONTENTS

## (CONTINUED)

SECTION		PAGE
5	TASK 5: SOFTWARE TEST AND INTEGRATION REQUIREMENTS . . . . .	5-1
5.1	TASK 5: SUMMARY . . . . .	5-1
5.2	STIL REQUIREMENTS SUMMARIZATION . . . . .	5-5
5.3	DEFINITION OF STIL OPERATIONAL MODES/DEVELOPMENT TOOLS . . . . .	5-11
5.4	STIL MODELING ANALYSIS . . . . .	5-45
5.5	STIL CONFIGURATION ANALYSIS . . . . .	5-61
5.6	STIL DEVELOPMENT PLAN ANALYSIS . . . . .	5-73
APPENDIX A:	FINAL REPORT: SOFTWARE DEVELOPMENT AND INTEGRATION PLAN	
APPENDIX B:	SPACELAB EXPERIMENT SOFTWARE FLOW (NASA PRESENTATION MAY 23, 1974)	
APPENDIX C:	SPACELAB SOFTWARE TEST AND INTEGRATION LABORATORY (NASA PRESENTATION MAY 23, 1974)	
APPENDIX D:	REFERENCE DATA	

## LIST OF FIGURES

Figure No.	Title	Page
1-1	Study Description . . . . .	1-3
1-2	Baseline CDMS Configuration . . . . .	1-13
1-3	Spacelab Software Logical Structure . . . . .	1-15/1-16
2-1	Study Approach for Definition of Spacelab Experiment Software Development Process . . . . .	2-3/2-4
2-2	Projected Experiment Flight Applications Software Modification . . . . .	2-10
2-3	Instructions to be Developed/Year for Experiment Flight Application Packages . . . . .	2-11
2-4	Spacelab Experiment Flight Application Delivery Profile . . . . .	2-12
2-5	PI to PI Flow . . . . .	2-15
2-6	Experiment Software Development Options . . . . .	2-18
2-7	PI Interfaces will Affect Experiment Development Concepts . . . . .	2-20
2-8	Overall Spacelab Flow . . . . .	2-24
2-9	CIS Payload Integration Time Line . . . . .	2-25
2-10	Experiment/Experiment Element Interface Validation and Checkout . . . . .	2-27
2-11	Development Philosophy Determination . . . . .	2-30
2-12	MSFC Software Development Timeline . . . . .	2-33
2-13	Launch and Prelaunch Timeline . . . . .	2-34
2-14	PI Software Development Timeline at his Facility . . . . .	2-35
2-15	Proposed Flight Application Software Set Characteristics . . . . .	2-38
2-16	Key Elements of Flight Application Software Characteristics . . . . .	2-39
2-17	Proposed Experiment Flight Application Software Development Process . . . . .	2-41
2-18	Experiment Flight Application Design Approach . . . . .	2-49
2-19	Typical Software Development Cycle with Supporting Development Tools . . . . .	2-55
2-20	Automatic System Build and Release Flow . . . . .	2-58
2-21	Impact on Software Development Facility on Testing . . . . .	2-59
2-22	Interpretive Computer Simulator Run Time Ratio/Flight Time Ratio . . . . .	2-60
3-1	Test and Checkout Software Concepts Study Approach . . . . .	3-2
3-2	Spacelab Engineering Model Flow . . . . .	3-7
3-3	Spacelab Hardware/Software Initial Unit Flow . . . . .	3-9
3-4	Spacelab Hardware/Software Operational Unit Flow . . . . .	3-11
3-5	Ground Checkout Software . . . . .	3-14
3-6	EGSE Software Role in Spacelab Test and Checkout . . . . .	3-18
3-7	On-Orbit Test and Checkout Software . . . . .	3-19
3-8	CIS Level III Testing Configuration . . . . .	3-22
3-9	CIS Level III Testing-Refurbishment of Support Section . . . . .	3-23
3-10	Level II Testing (KSC) . . . . .	3-24
3-11	Level I Testing (KSC) . . . . .	3-26
3-12	Functional Representation of CDMS Software Design Approach . . . . .	3-30
4-1	Spacelab POC and Preprocessing Baseline . . . . .	4-5
4-2	Spacelab Data Flow . . . . .	4-10
4-3	Mission Planning Process . . . . .	4-12
4-4	Baseline Mission Planning Systems Architecture . . . . .	4-14
4-5	PI Mission Support Concepts, Options 1 and 2 . . . . .	4-19
4-6	PI Mission Support Concepts, Options 3 and 4 . . . . .	4-20

## LIST OF FIGURES (CONT'D)

Figure No.	Title	Page
5-1	Software Test and Integration Task Flow . . . . .	5-2
5-2	STIL Operational Modes and Development Tools . . . . .	5-8
5-3	Operational Modes and Associated Software Development Tools . . . . .	5-12
5-4	Functional Diagram of Realtime Testing Configuration . . . . .	5-14
5-5	STIL Realtime Individual Computer Simulation . . . . .	5-18
5-6	CID Control and Test Utilization . . . . .	5-20
5-7	Functional Simulation Description . . . . .	5-24
5-8	Functional Description of ICS in Batch Environment . . . . .	5-26
5-9	Functional Representation of Data Reduction Software . . . . .	5-29
5-10	STIL Host Machine Operating System Concept . . . . .	5-31
5-11	Proposed Software Management System Flow . . . . .	5-34
5-12	Source Data Management . . . . .	5-36
5-13	Configuration Management System . . . . .	5-38
5-14	Automatic Release System . . . . .	5-39
5-15	Spacelab Software Scheduling System . . . . .	5-40
5-16	Preliminary STIL Configuration Model Task Flow . . . . .	5-46
5-17	Throughput as a Function of CPU Capability . . . . .	5-51
5-18	Throughput as a Function of Core Memory for 1 MIPS CPU . . . . .	5-53
5-19	Marginal CPU Capability Relative to Memory Allocation . . . . .	5-54
5-20	Impact of Insufficient I/O Capability . . . . .	5-56
5-21	Growth Computer Profiles . . . . .	5-57
5-22	Sensitivity Analysis . . . . .	5-58
5-23	Selection of Optimal CPU Memory Size Combinations . . . . .	5-60
5-24	STIL Functional Diagram . . . . .	5-63
5-25	CID/CDMS Interface . . . . .	5-65
5-26	Centralized Single CPU Configuration . . . . .	5-67
5-27	Decentralized CPU Configuration with Dedicated Realtime Processor . . . . .	5-68
5-28	Decentralized CPU Configuration with Dedicated I/O Processor . . . . .	5-70
5-29	Decentralized CPU Configuration with Expansion for other Spacelab Functions . . . . .	5-71/5-72
5-30	Preliminary STIL Software/Hardware Development Plan . . . . .	5-74
5-31	Phase I - Preliminary Supportive and Batch Mode Development Plan . . . . .	5-76
5-32	Phase II - Preliminary Realtime Mode Development Plan . . . . .	5-77/5-78

## LIST OF TABLES

Table No.	Title	Page
1.1	Summary of Study Conclusions . . . . .	1-6
2.1	Spacelab Traffic Model Summary . . . . .	2-6
2.2	Spacelab Payload Software Sizing Table . . . . .	2-8
2.3	PI Participation in Experiment Software Development Tasks . . . . .	2-16
2.4	Software Functions Required by a Representative Group of Experiments . . . . .	2-31
2.5	Software Development Options . . . . .	2-36
2.6	Proposed Software Development Responsibilities . . . . .	2-44
2.7	Development Tools Required for Spacelab Onboard Software . . . . .	2-54
2.8	Skylab/Spacelab Program Similarities . . . . .	2-64
2.9	Summary of Software Management Load on STIL . . . . .	2-67
2.10	Summary of Software Implementation Requirements on STIL . . . . .	2-69
2.11	Summary of Software Verification on STIL . . . . .	2-71
2.12	Summary of Software Integration Load on STIL . . . . .	2-72
2.13	Summary of STIL Requirements from Flight Application Development . . . . .	2-73/2-74
3.1	Summary of Spacelab Ground Operations Analysis . . . . .	3-12
3.2	Size of Test and Checkout Software to be Maintained by NASA . . . . .	3-33
3.3	Summary of Test and Checkout Impact on STIL . . . . .	3-39
5.1	STIL Support Load per Day . . . . .	5-7
5.2	Operating System Services . . . . .	5-33
5.3	Preliminary STIL Data Baseline . . . . .	5-41
5.4	STIL Load Requirements . . . . .	5-43/5-44
5.5	STIL Job Frequencies . . . . .	5-49
5.6	STIL Model Input Data Summary . . . . .	5-50

# STUDY SYNOPSIS

1

SECTION		PAGE
1	STUDY SYNOPSIS. . . . .	1-1
1.1	STUDY INTRODUCTION . . . . .	1-1
1.1.1	Study Objectives . . . . .	1-1
1.1.2	Study Plan . . . . .	1-2
1.1.3	Final Report Format . . . . .	1-2
1.2	MAJOR STUDY CONCLUSIONS . . . . .	1-5
1.2.1	Development/Delivery/Maintenance Conclusions . . . . .	1-5
1.2.2	STIL Related Conclusions . . . . .	1-7
1.2.3	NASA's Role in CDMS/EGSE Software . . . . .	1-8
1.3	STUDY RECOMMENDATIONS . . . . .	1-9
1.3.1	STIL Definition . . . . .	1-9
1.3.2	Spacelab Software Documentation Tree . . . . .	1-9
1.3.3	Principal Investigator's Software Designers' Document . . . . .	1-9
1.3.4	Define Interface/Interaction Between Spacelab Support Facilities . . . . .	1-10
1.3.5	Define Software Integration Procedures . . . . .	1-10
1.3.6	Initial Specification of Data Reduction Software . . . . .	1-10
1.3.7	Detail Mission Operations & Crew Training Software . . . . .	1-10
1.4	STUDY BASELINE . . . . .	1-11
1.4.1	Study/Assumptions . . . . .	1-11
1.4.2	Spacelab Ground Operations and Facilities . . . . .	1-11
1.4.3	Spacelab CDMS Hardware Configuration . . . . .	1-12
1.4.4	Spacelab Software Configuration . . . . .	1-14

# STUDY SYNOPSIS 1

## 1.1 STUDY INTRODUCTION

The Spacelab will consist of integrated, reusable modules which will be transported to and from orbit by the Space Shuttle. It will reside in the Space Shuttle payload bay throughout its mission and will consist of a pressurized module and pallet segments. The Spacelab will support experiments for scientific and applications research and will have a life span of ten years or fifty missions with mission duration extendable up to thirty days.

The Spacelab will be a development of the multinational efforts of the European Space Research Organization (ESRO), with technical support provided by the National Aeronautics and Space Administration (NASA). The Spacelab will be defined by ESRO, designed and built in Europe, and then delivered to the United States to be integrated with the experiments. The experiments which will be carried into orbit in the Spacelab will be developed and built by the universities and principal investigators from both the United States and Europe. NASA will assume operational responsibility after accepted delivery of the first two flight units from ESRO.

Because of the unique characteristics of Spacelab and the Space Shuttle program, software will play a major role because of its flexibility and ability to be rapidly modified to meet changing requirements. All software subsystems which comprise the Spacelab software and provide for its operation must be comprehensible to those who will use it as well as those who will design and implement it. Comprehension must be easily attained by those who would understand Spacelab's operation, use, and integration with the Space Shuttle and by those conducting acceptance tests, checkout, maintenance, repair, and modification. Because of this widespread necessity for comprehensibility within the Spacelab project, it is essential that particular emphasis be placed on planning and procedures.

This final report concludes the study of Spacelab Software Development and Integration Concepts authorized by the National Aeronautics and Space Administration, Marshall Space Flight Center, Contracts NAS8-30376 and NAS8-30538. Contract NAS8-30538 was performed in the March through August 1974 time frame and was a continuation of an initial study effort. The output of this contract consists of the following: Final Report, Spacelab Software Management Plan (Appendix A); Spacelab Experiment Software Flow, NASA presentation May 23, 1974 (Appendix B); Spacelab Software Test and Integration Laboratory, NASA presentation May 23, 1974 (Appendix C); and Reference Data (Appendix D).

### 1.1.1 STUDY OBJECTIVES

The objective of this study effort was to provide NASA an insight into the complexity and magnitude of the Spacelab software challenge. During the study, IBM assessed the current Spacelab program concepts, anticipated flight schedules, and ground operation plans. From this assessment, those areas requiring immediate attention were identified. The study data contained with-

in this final report provides sufficient information for NASA's use in subsequent Spacelab software activity.

The study was primarily directed toward identifying and solving problems related to the Experiment Flight Application and Test and Checkout software executing in the Spacelab Onboard Command and Data Management Subsystem (CDMS) computers and Electrical Ground Support Equipment (EGSE) computer. Additionally, a cursory examination of the required mission operation software was conducted to provide a global understanding of interrelationships of the Spacelab operation software environment.

The study provides a conceptual base from which NASA can proceed into the development phase of the Software Test and Integration Laboratory (STIL) and establishes guidelines for definition of standards which will ensure that the "total" Spacelab software is understood prior to entering development. Proposed software concepts and guidelines have been developed which will provide NASA with the necessary information to effectively interface with ESRO. Additionally, they ensure effective utilization of the software to be developed by ESRO for use during the Spacelab operational phase.

#### 1.1.2 STUDY PLAN

The study was divided into five disciplined tasks and each task broken into study elements to provide a means of systematically addressing and documenting the results of the study. This approach also provided the necessary NASA management visibility to redirect study efforts to keep abreast of the changing Spacelab environment during the study period.

As illustrated in Figure 1-1, the study effort was accomplished over two contract periods and was originally begun in September 1973. Because of the interrelationship among the tasks, they were worked in a parallel manner; however, each resulted in individual outputs and is documented separately in the final report. Due to the interest generated in the May midterm presentation, extensive effort during the latter portion of the study was directed toward bounding the STIL processing load so that NASA can move confidently into a final STIL definition and development activity.

#### 1.1.3 FINAL REPORT SUMMARY

The final report format was designed to be informal in nature but structured to follow the study task efforts. It is difficult to effectively record the full activities of a study contract which addresses conceptual ideas as this one does; however, it is the intent to provide sufficient insight into each task and element so that the reader can understand the rationale utilized in deriving conclusions and concepts. Each element provides the theme, conclusions derived, and a discussion supporting the conclusions or defining the concept developed.

The following paragraphs briefly summarize the content of each section of this report.



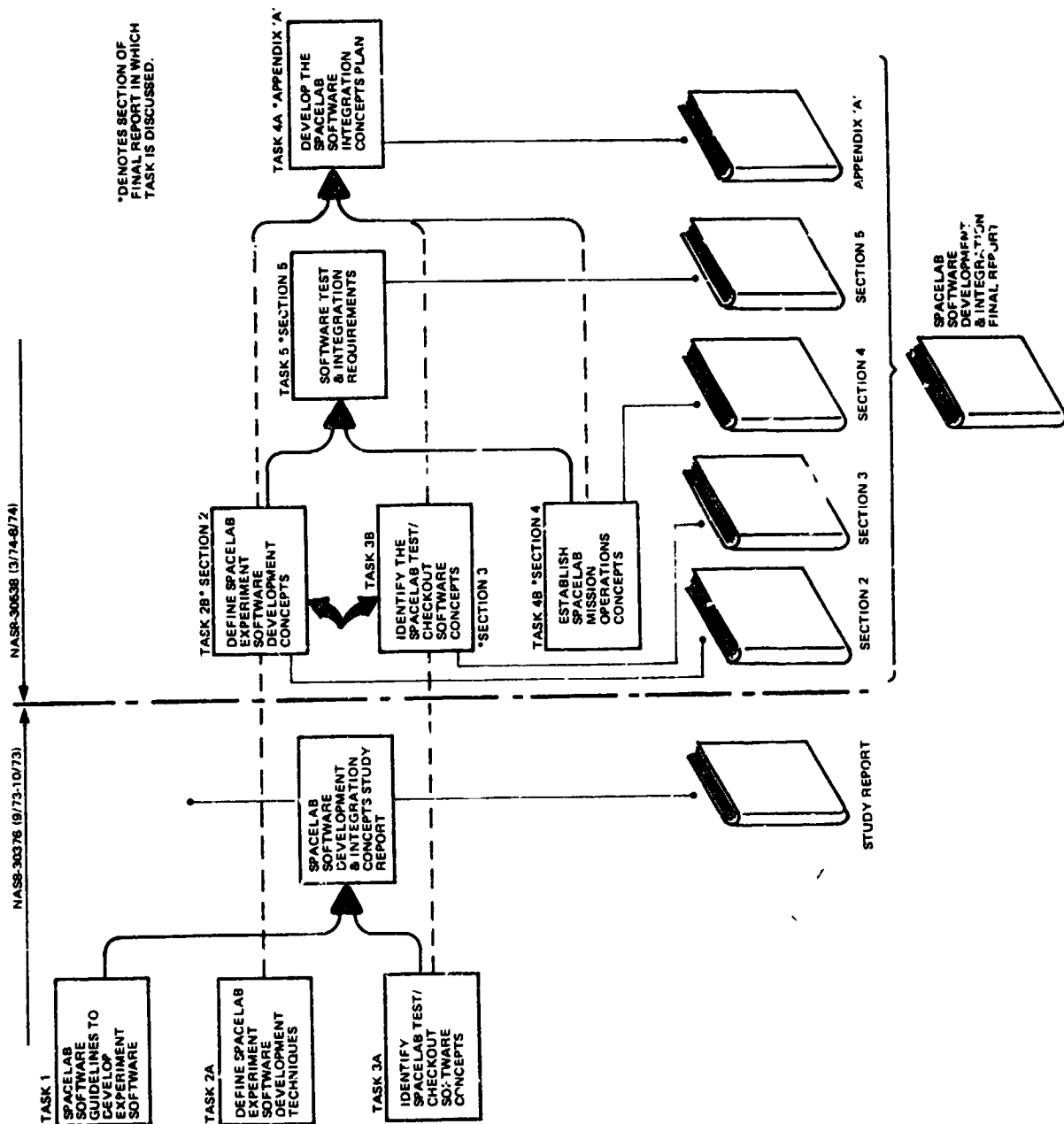


Figure 1-1. Study Perspective

## Section 1

Section 1 establishes the overall study concept and summarizes the major study conclusions, makes recommendations for future effort, and documents the general study baseline.

## Section 2

The results of Task 2B - Definition of Experiment Software Development Concepts - is documented in Section 2. Task 2B provides insight into problems associated with the development, integration, and testing of Experiment Flight Applications software. The discussion provides a development concept which circumvents anticipated problem areas and identifies the requirements and impact on the STIL.

## Section 3

Task 3B - Identification of Spacelab Test and Checkout Software Concepts - is documented in Section 3. Task 3B addresses the requirements for Test and Checkout Software within the Spacelab integration testing plan and discusses the software concepts to be utilized in satisfying those requirements.

## Section 4

Section 4 addresses Spacelab mission operation requirements and identifies the interaction between the Spacelab onboard software and the mission operations software. Functional interfaces with the Payload Operations Center and the Preprocessing Center are identified.

## Section 5

Task 5 - Software Test and Integration Requirements - addresses the requirements of the STIL, establishes a preliminary development plan, and provides an analysis of anticipated loads on the STIL for both the first two missions and the operational phase.

## Appendix A

The Spacelab Software Development and Integration Plan has been developed as a baseline for MSFC to proceed with the Spacelab software development. This plan completes the documentation of Task 4 - Develop the Spacelab Integration Concepts Plan.

## Appendices B and C

Appendices B and C contain presentation material which was presented to MSFC during the study activity.

## Appendix D

Appendix D contains reference data utilized in the performance of this study. This data consists of:

- List of References
- List of Acronyms

## 1.2 MAJOR STUDY CONCLUSIONS

Although the conclusions established during the conduct of the study are included within appropriate report sections, the key conclusions which are of primary importance to NASA have been summarized in Table 1.1 and are discussed in the following paragraphs.

### 1.2.1 DEVELOPMENT/DELIVERY/MAINTENANCE CONCLUSIONS

CONCLUSION: *DEVELOPMENT AND DELIVERY OF 416,520 INSTRUCTIONS FOR EXPERIMENT FLIGHT APPLICATIONS SOFTWARE WILL BE REQUIRED IN 1981.*

DISCUSSION: Based on the Space Shuttle traffic model assessment conducted during the study, the maximum development activity will occur in 1981. Within that year, development and delivery of Experiment Flight Applications software for eight new flight sets and 10 reflight flight sets, requiring development of 416,520 instructions, must be accomplished. The capability to support the maximum development load early in the Spacelab program will require that facilities and manpower requirements be addressed in a systematic manner to ensure availability on the need-dates with minimum phase-up time. This is significantly different from previous NASA space programs which had phase-up time to develop and test operational capabilities in a systematic manner.

CONCLUSION: *TEST AND CHECKOUT SOFTWARE MAINTENANCE BY NASA WILL REQUIRE SUPPORT OF 301,700 INSTRUCTIONS/610 MODULES.*

DISCUSSION: The Test and Checkout software sets, to be developed by ESRO and provided to NASA for establishment of the maintenance baseline, have been estimated to consist of approximately 301,700 instructions. These instructions will be organized into approximately 610 program modules. To provide the capability to support and maintain software systems of this magnitude, trained manpower as well as development tools/facilities will be required.

The NASA STIL must support the capabilities for test and checkout software and maintenance.

CONCLUSION: *SOPHISTICATED SOFTWARE MANAGEMENT TECHNIQUES WILL BE REQUIRED*

DISCUSSION: In an environment in which up to 18 Experiment Applications software packages can be simultaneously undergoing development, strict software management control will be required. This software management system must be terminal-oriented and provide the capabilities of source maintenance, configuration management, and automatic release. In addition to use in the management of Experiment Flight Applications software, the same techniques will be utilized for subsystem and EGSE software.

*Table 1.1. Summary of Study Conclusions*

**DEVELOPMENT/DELIVERY/MAINTENANCE**

- Development and delivery of 416,520 instructions for Experiment Flight Applications Software will be required in 1981.
- Test and Checkout Software maintenance by NASA will require support of 301,700 instructions/610 modules.
- Sophisticated Software Management techniques will be required during development and maintenance of software.
- The development concept for Experiment Flight Applications Software must support five development options.

**STIL RELATED**

- A Software Test and Integration Laboratory (STIL) will be required in support of CDMS and EGSE software development.
- The STIL must be operational prior to first quarter of 1978.

**NASA'S ROLE IN CDMS/EGSE SOFTWARE**

- NASA will be responsible for integration, verification, and delivery of CDMS/EGSE software.

CONCLUSION: *DEVELOPMENT CONCEPT FOR EXPERIMENT FLIGHT APPLICATIONS SOFTWARE MUST SUPPORT FIVE MAJOR DEVELOPMENT OPTIONS.*

DISCUSSION: The development concept established during the study for Experiment Flight Applications software provided the capabilities to support the following development options:

Option 1 - PI develops the package on the NASA STIL.

Option 2 - NASA/STIL team develops the package.

Option 3 - PI develops the package on a copy of the STIL.

Option 4 - PI develops the package on his STIL compatible computer and NASA provides a CDMS simulator.

Option 5 - PI develops the package on his own computer and NASA supplies a CDMS simulator.

#### 1.2.2 STIL RELATED CONCLUSIONS

CONCLUSION: *TEST AND INTEGRATION LABORATORY (STIL) WILL BE OPERATIONAL PRIOR TO EXPERIMENT FLIGHT APPLICATIONS SOFTWARE DEVELOPMENT AND CHECKOUT SOFTWARE MAINTENANCE.*

DISCUSSION: As performed within the study established that a dedicated facility for use in CDMS and EGSE software development and maintenance must be provided by NASA. The maximum load on this facility was determined to be approximately 350 runs per day. These runs will consist of realtime simulations, batch processing, and supportive functions needed in providing the required software development environment. It was also determined that the STIL must contain a CDMS and must provide a realtime simulation capability. Modeling of the STIL load with a GPSS model established that the optimum host computer must have a CPU capability of approximately 3 Million Instructions Per Second (MIPS) and a memory capacity of approximately 3 million bytes and must provide the software development tools and services required for CDMS/EGSE software development and maintenance. The supportive functions such as software management and data base will also be provided by the STIL.

CONCLUSION: *THE STIL MUST BE OPERATIONAL PRIOR TO THE 1ST QUARTER OF 1978.*

DISCUSSION: The ability to support the delivery of the Engineering Model (EM) software in the first quarter of 1978 will require that the STIL be operational prior to that time. In order to meet this need-date for operational utilization of the STIL, a preliminary development plan was developed. This plan indicates that the development phase for the STIL should begin in early 1975.

### 1.2.3 NASA'S ROLE IN CDMS/EGSE SOFTWARE

CONCLUSION: NASA MUST BE RESPONSIBLE FOR INTEGRATION, VERIFICATION, AND DELIVERY OF CDMS/EGSE SOFTWARE.

DISCUSSION: Because of the severe time constraints on Spacelab integration testing at all levels, it will be required that verification testing of all software be accomplished prior to delivery. This verification must ensure that hardware/software incompatibilities are minimized, or the overall launch schedules can be impacted. NASA must perform this verification under strict configuration control at the STIL. As a result, each PI developing his application package must adhere to strict interface standards to ensure that integration and verification will proceed in an orderly manner.

In support of the integration function, the software design must ensure separation of independent applications and must provide an operating system to control execution of those application packages.

### 1.3 STUDY RECOMMENDATIONS

As a result of the analyses performed during this study, additional areas of investigation have been identified for subsequent study. The following paragraphs indicate the principal areas of future study and discuss briefly the type of study activity to be performed.

#### 1.3.1 STIL DEFINITION

As established during the study, development of the STIL should begin during 1975. To accomplish this schedule, a detailed definition of the STIL hardware and software should be performed prior to initiation of the development activity. Sections 2, 3, and 5 of this report will provide the data base required to proceed with the STIL definition phase. The results of this study would define the host computer, the CDMS Interface Device (CID), and required software in sufficient detail to allow procurement of the STIL hardware and software.

#### 1.3.2 SPACELAB SOFTWARE DOCUMENTATION TREE

During this study, it was determined that there exists a requirement to develop an integrated software documentation tree for all Spacelab related software. This tree should provide interrelationships and interfaces among the various software development areas (i.e., ESRO, Payload Operations Center, Preprocessing Center, Shuttle Mission Operations Center, STIL, etc.). Also included within this task should be a recommended format and level of content for each document.

#### 1.3.3 PRINCIPAL INVESTIGATORS SOFTWARE DESIGNER'S DOCUMENT

As described in Paragraph 2.6 of this document, a PI Software Designer's Document is the single source of data which defines the NASA services to the experiment application programmer and/or STIL user and defines "rules" which must be followed in development of Experiment Flight Applications software. This document must be a living document throughout the Spacelab life cycle to meet the changing needs of the PI. Utilizing the concepts defined during this study, the baseline document can be designed and an initial publication produced. Examples of the contents of this document are:

##### Services Provided by CDMS

- Hardware interface rules between experiment and CDMS
- Software interface rules between Operating System and Applications software
- CDMS hardware/software performance parameters

#### Services Provided by STIL

- Software development tools
- Software test tools
- Procedures for using STIL

#### Software Management Requirements

- Software development philosophies
- Software standards
- Configuration management procedures

#### 1.3.4 DEFINE INTERFACE/INTERACTION BETWEEN SPACELAB SUPPORT FACILITIES

Data base interaction among the various Spacelab facilities will be a major problem and concern. Proper emphasis must be placed on assuring that the STIL data base is interrelated with other major data bases such as mission planning, crew training, Payload Operations Center, etc. This common sharing of uniform data is mandatory to a cost effective overall software development.

#### 1.3.5 DEFINE SOFTWARE INTEGRATION PROCEDURES

The most cost effective methods of putting Spacelab software on-line is to utilize existing software modules, packages, and sets (with minimum modifications) to meet the requirements of Spacelab. ESRO will initially develop major Spacelab software sets. The Apollo, Saturn, Skylab, and Shuttle programs have developed a significant amount of software which can be applied to the facilities which will be supporting Spacelab operational phases. An integration plan must be developed which will provide procedures for an orderly integration and verification of available software into the major Spacelab support facilities (STIL, Mission Operations, Crew Training, CIS).

#### 1.3.6 INITIAL SPECIFICATION OF DATA REDUCTION SOFTWARE

Skylab has proven that massive amounts of data must be processed to reap the full benefits of a scientific mission. The initial requirements for NASA's Spacelab data reduction capability should be documented and baselined as well as those requirements for detailed data analysis on NASA sponsored experiments. Present concepts call for the PI to provide all data analysis software; however, the majority of projected PIs are also NASA employees. This will require that NASA provide data analysis software for its own PI.

#### 1.3.7 DETAIL MISSION OPERATIONS AND CREW TRAINING SOFTWARE

These two important aspects of Spacelab software must be defined and documented so that the impact of their requirements on CDMS and STIL software development schedules and requirements can be determined.



#### 1.4 STUDY BASELINE

A baseline understanding of the Spacelab operational and development environments was established early in the study. This baseline was modified as the study progressed due to the publication of the Spacelab Ground Operations Plan (Item 1, List of References), ERNO Proposal (Item 27, List of References), ESRO MSFC July briefing, and continued analysis of the software development concepts. The following paragraphs are representative of the current baseline.

##### 1.4.1 STUDY ASSUMPTIONS

Every study must establish a set of assumptions and limitations to guide the study effort to a successful conclusion with meaningful results. The following list denotes the major assumptions made within this study.

- Prime study emphasis was placed on Experiment CDMS software development and integration concepts and on establishment of a set of baseline STIL requirements.
- ESRO will make available to NASA the following software to be used in the STIL.
  - CDMS and EGSE compilers/assemblers/linkage editors
  - Common CDMS Operating System
  - Subsystem CDMS Flight Applications
  - EGSE Operating System
  - EM, Flight Unit 1 (FU1) and FU2 Test and Checkout software
- NASA will be responsible for all Spacelab software following acceptance of Flight Unit 2.

##### 1.4.2 SPACELAB GROUND OPERATIONS AND FACILITIES

A major portion of the software concept analysis utilized the Spacelab Ground Operations Plan and projected flight operations plans. This analysis established the following baseline data:

- The Software Test and Integration Laboratory (STIL) - will provide the total computational capabilities for maintenance of all CDMS and EGSE computer software. It will have the capability for software development activities including design, development, integration, and validation of Experiment CDMS Applications software. Complete software management tools and data base will be maintained at the STIL.

- The Central Integration Site (CIS) - will be utilized to perform final hardware/software integration and validation.
- The CIS EGSE (Core Segment/Subsystem Simulator) - will have sufficient simulation facility to perform hardware/software integration and validation of the Experiment CDMS Computer software.
- The Engineering Model - will be utilized as a CIS hardware/software integration tool and for utilization in testing of major modifications of the subsystem CDMS software.
- The Spacelab Subsystem and Experiments - will be designed to be test compatible in that software will be able to detect faults and isolate to the LRU level.
- The Launch and Landing Site Facilities - will be users of the software systems and will not be involved in CDMS and EGSE software development process.
- The Payload Operations Center - will be utilized to provide ground monitoring and ground PI interface to the CDMS.
- The Preprocessing Facility - will be utilized to perform data reduction on Spacelab data. The processing will consist of formatting all data into a common compressed format. The PI will be responsible for all data analysis.

#### 1.4.3 SPACELAB CDMS HARDWARE CONFIGURATION

The CDMS hardware configuration was evolving during the study period, and several configurations were utilized during the study; however, the final configuration was the ESTEC baseline presented at the July MSFC briefing. This baseline is represented in Figure 1-2, and the following are key points relative to the configuration which influenced the study direction.

- CDMS computers are all identical with perhaps the backup and experiment computers having a larger core memory.
- Experiment and subsystem software can be designed and tested independently. Backup computer is powered off prior to utilization. Upon power on, the backup computer software will be initialized via the orbiter interface from either the orbiter computer, PSS station, or POC.
- Mass memory is to be implemented as a read only device.
- EGSE computer interface will be similar to or perhaps the same as the orbiter interface to the CDMS computers.
- EGSE computer is expected to be similar in design to the CDMS computers.

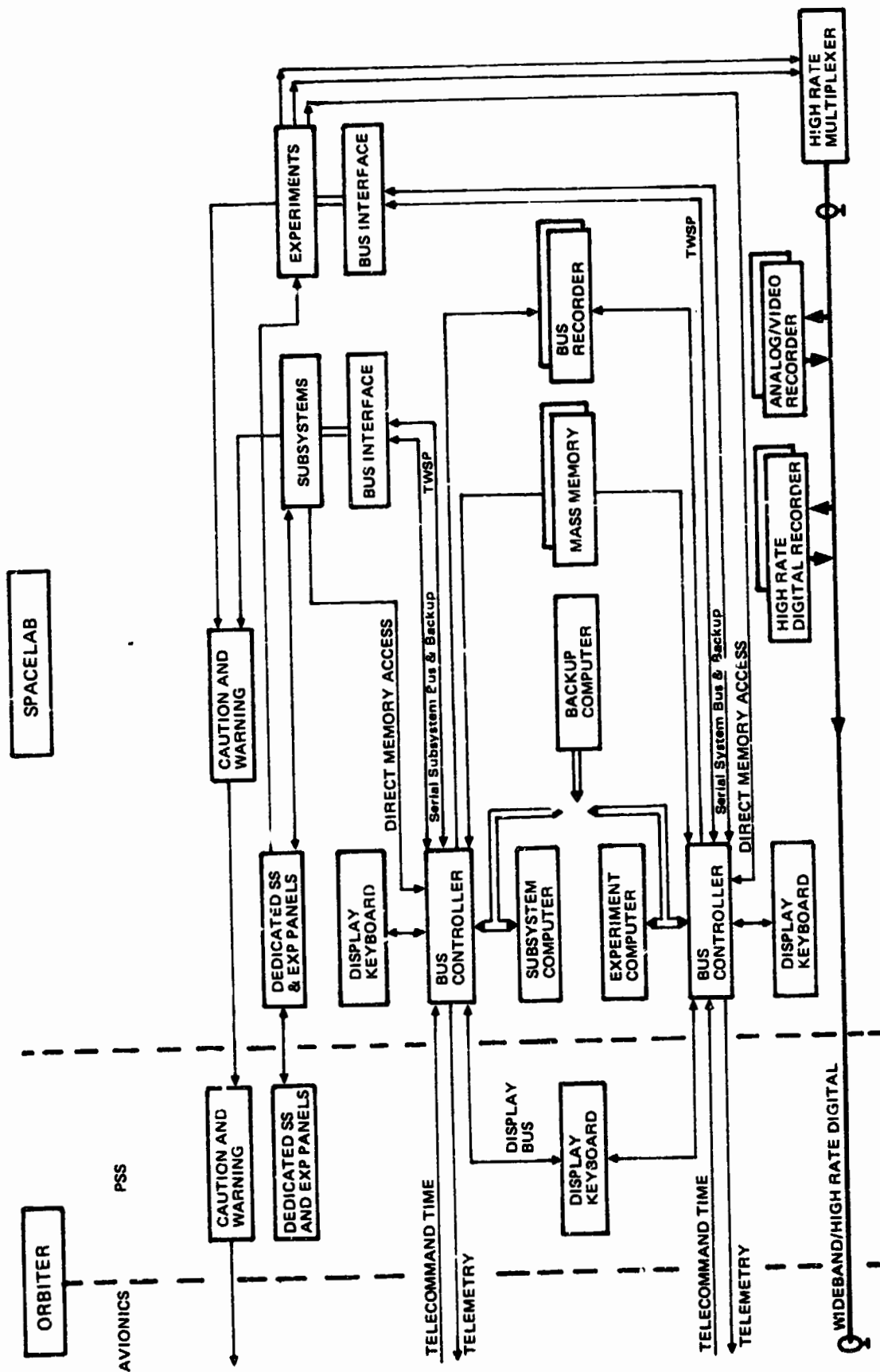


Figure 1-2. Baseline CDMS Configuration

#### 1.4.4 SPACELAB SOFTWARE CONFIGURATION

It is premature at this time to establish a fixed software architecture for the CDMS and EGSE software; however, its basic hierarchy and interrelationship can be established. To avoid confusion due to terminology, the software definition and structure utilized by ESTEC has been incorporated within this report.

For the purposes of this study, the hierarchical relationships as depicted in Figure 1-3 have been used. The relationships (from the lowest identified element to the highest) can be summarized as follows:

- Module - is the lowest element of software to be under configuration control. The module is considered to contain 100 HOL program statements and is considered as a building block. The module can be considered identical to a circuit board or chip in hardware.
- Package - is a combination of modules into a logical unit to satisfy the requirements of a particular function. An example of a package is those modules which make up a flight applications function. In hardware, this would be analogous to a subsystem.
- Set - is a combination of packages to satisfy the requirements of a payload. An example of a set would be the combination of flight applications and operating system packages to form the software to execute in the experiment CDMS computer while in orbit. In hardware, this would be referred to as a major subsystem. Sets can be combined to form sets of sets as, in a hardware sense, subsystems together form systems.

As can be seen in Figure 1-3, the Mission Set is composed of the CDMS Flight Set, the CDMS Ground Checkout Set, and the EGSE Ground Checkout Set. Each CDMS Set, correspondingly, is composed of an Experiment CDMS and Subsystem CDMS Set. The figure thus establishes the logical relationships among modules, packages, sets, and sets of sets.



# TASK 2B: DEFINITION OF SPACELAB EXPERIMENT DEVELOPMENT CONCEPTS

2

SECTION		PAGE
2	TASK 2B: DEFINITION OF SPACELAB EXPERIMENT DEVELOPMENT CONCEPTS	2-1
2.1	TASK 2B: SUMMARY	2-1
2.1.1	Task Objective	2-1
2.1.2	Task Conclusions	2-1
2.1.3	Task 2B Study Approach	2-2
2.2	TRAFFIC MODEL & SCHEDULE ASSUMPTIONS ANALYSES	2-5
2.2.1	Theme	2-5
2.2.2	Conclusions	2-5
2.2.3	Discussion	2-5
2.2.3.1	Software Development Analysis	2-7
2.2.3.2	Software Delivery Analysis	2-12
2.3	PI TO PI FLOW ASSESSMENT	2-13
2.3.1	Theme	2-13
2.3.2	Conclusions	2-13
2.3.3	Description	2-13
2.3.3.1	Classes of PIs	2-13
2.3.3.2	PI to PI Flow	2-14
2.3.3.3	PI Requirements on Experiment Development Concepts	2-17
2.4	PI/CDMS SOFTWARE INTERFACES	2-19
2.4.1	Theme	2-19
2.4.2	Conclusions	2-19
2.4.3	Discussion	2-19
2.4.3.1	Interactive Control Requirements	2-19
2.4.3.2	Interactive Software Requirements	2-22
2.4.3.3	Interactive Language Requirements	2-22
2.5	EXPERIMENT CHECKOUT/VERIFICATION TECHNIQUES	2-23
2.5.1	Theme	2-23
2.5.2	Conclusions	2-23
2.5.3	Discussion	2-23
2.5.3.1	Permission Timeline	2-23
2.5.3.2	Hardware/Software Validation	2-26
2.5.3.3	Software Testing Prior to CIS Delivery	2-26
2.5.3.4	Experiment Testing Prior to CIS Delivery	2-28
2.6	DEVELOPMENT PHILOSOPHY DETERMINATION	2-29
2.6.1	Theme	2-29
2.6.2	Conclusions	2-29
2.6.3	Discussion	2-29
2.6.3.1	Development Philosophy Considerations	2-29
2.6.3.2	Experiment Flight Applications Software Characteristics	2-37
2.6.3.3	Experiment Flight Applications Software Development Process	2-37
2.6.3.4	Software Development Responsibilities	2-43

## (CONTINUED)

SECTION		PAGE
2	(CONTINUED)	
2.7	EXPERIMENT SOFTWARE REQUIREMENTS . . . . .	2-47
2.7.1	Theme . . . . .	2-47
2.7.2	Conclusions . . . . .	2-47
2.7.3	Discussion . . . . .	2-47
2.7.3.1	Design Requirements . . . . .	2-47
2.7.3.2	Language Requirements . . . . .	2-50
2.7.3.3	Operating System Requirements . . . . .	2-50
2.7.3.4	Experiment Applications Packages . . . . .	2-51
2.8	DEVELOPMENT TOOLS REQUIREMENTS . . . . .	2-53
2.8.1	Theme . . . . .	2-53
2.8.2	Conclusions . . . . .	2-53
2.8.3	Discussion . . . . .	2-53
2.8.3.1	Development Tools . . . . .	2-53
2.8.3.2	Development Facilities . . . . .	2-57
2.9	STIL REQUIREMENTS . . . . .	2-63
2.9.1	Theme . . . . .	2-63
2.9.2	Conclusions . . . . .	2-63
2.9.3	Discussion . . . . .	2-63
2.9.3.1	STIL Requirements for Experiment Flight Applications	
	Software . . . . .	2-65
2.9.3.2	Summary of STIL Requirements . . . . .	2-72

## TASK 2B: DEFINITION OF SPACELAB EXPERIMENT SOFTWARE DEVELOPMENT CONCEPTS 2

This section of the Spacelab Software Development and Integration Concepts Final Report describes the analysis performed in establishing a recommended development concept for Experiment Flight Applications (EFA) software and the resulting development tools and facilities needed to support that concept.

### 2.1 TASK 2B: SUMMARY

The EFA software, to be developed in support of the Spacelab missions, is a critical element in achieving the scientific objective of the program. Because of the significant number of missions to be supported with a wide variety of experiments on each mission, coupled with an extremely ambitious launch schedule, it is obvious that a new software development environment will exist. Previous NASA experience in space software development has been characterized by long lead time in development with a slowly evolving software baseline developing as the program matures. Such a development environment will not exist for Spacelab experiments because of the rapidly changing mission objectives; therefore, improved development concepts must be established to meet the challenges of the Spacelab experiment software development requirements.

#### 2.1.1. TASK OBJECTIVE

The primary objective of the study task was to define and recommend a development concept for Experiment Flight Applications software. The Experiment Flight Applications software is defined as the onboard software packages which provide processing services for each unique experiment. In achieving this objective, the primary considerations were definition of development environment, establishment of development concepts, and identification of impact of development concept.

#### 2.1.2 TASK CONCLUSIONS

Subtask conclusions are presented with each subsequent subsection of this document as appropriate. The following conclusions are those which IBM feels are significant and are highlighted for the reader's convenience.

- The proposed development concept described in Paragraph 2.6 will support all options of Experiment Flight Applications software development.
- A Spacelab dedicated Software Test and Integration Laboratory (STIL) is necessary to support the EFA development concept, traffic model, and projected software development load.
- If additional STILs are required they should all be hardware/software compatible.



### 2.1.3 TASK 2B STUDY APPROACH

IBM's approach in accomplishing the Definition of Spacelab Experiment Software Development Concepts study task is illustrated in Figure 2-1. The first phase (Identification of Development Requirements) addressed the Space-lab environment and philosophy requirements. This provided the requirements which must be supported by the flight applications software development concepts. The second phase (Concepts Determination) identified the possible options which could be utilized in satisfying the requirements, evaluated those options, and proposed the Experiment Flight Applications Development concept. The third phase (Areas of Impact of Concepts) established those requirements on flight applications software, development tools, and development facilities which result from the development concepts.

The numbers appearing in the figure indicate the subsequent paragraphs in which the detailed analysis is discussed. Within each paragraph the conclusions, established by that specific analysis, are identified and discussed.

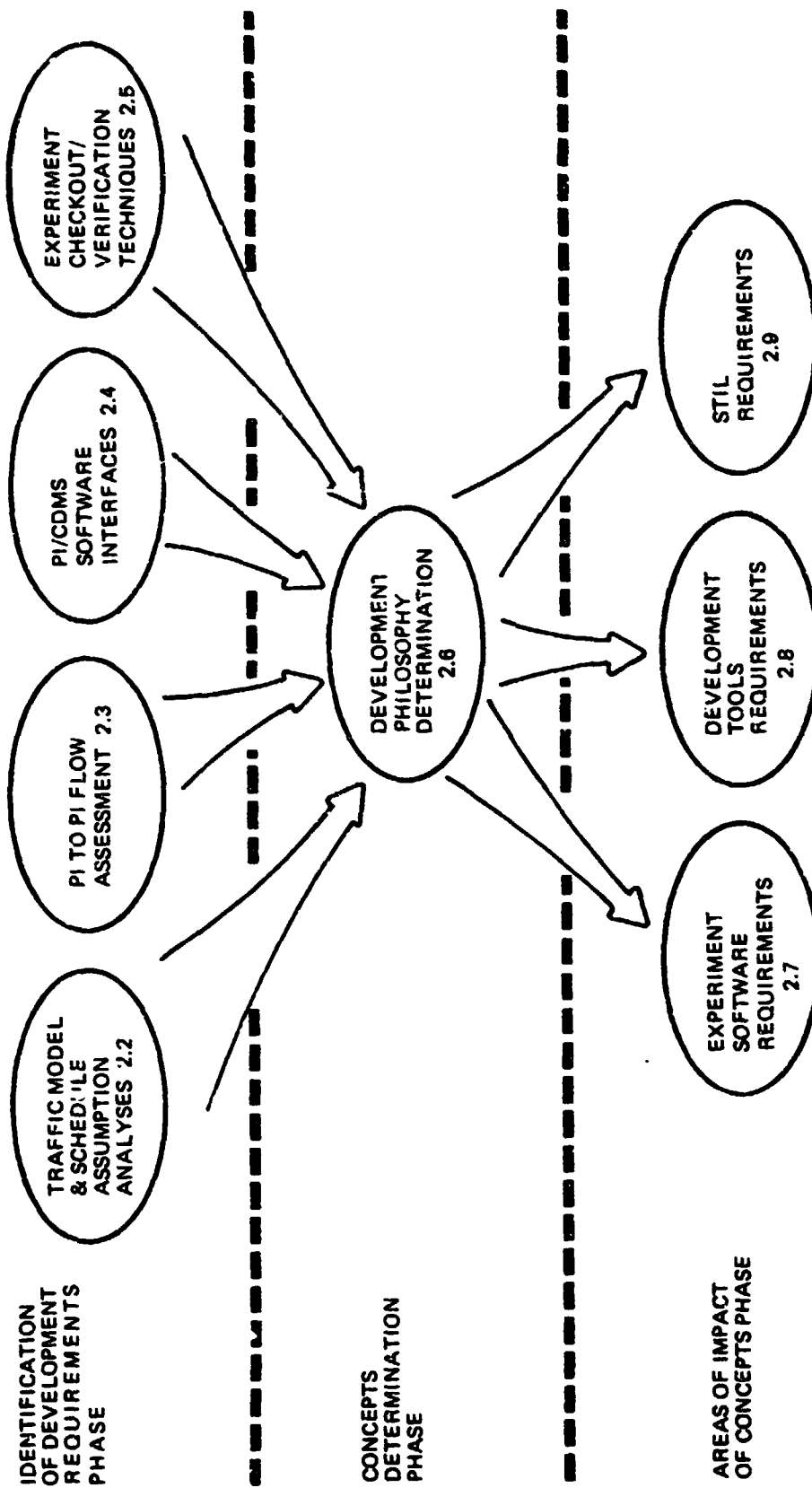


Figure 2-1. Study Approach for Definition of SpaceLab Experiment Software Development Concepts

## 2.2 TRAFFIC MODEL AND SCHEDULE ASSUMPTIONS ANALYSES

### 2.2.1 THEME

In establishing the development concept for Spacelab Experiment Flight Applications (EFA) software, the operational constraints, within which the concept must be structured, must be fully understood. To achieve this understanding, IBM evaluated the October 1973 Space Shuttle Traffic Model (Item 24-List of References) and other appropriate Spacelab related studies. Through this evaluation process, the EFA operational environment was established.

### 2.2.2 CONCLUSIONS

The results of the analysis of the traffic model have established the following conclusions:

- The peak development activity year for EFA software will be 1981. This will require a rapid NASA buildup of software support capability in order to support the development burden.
- A maximum of 36 NASA EFA software packages will be undergoing development in 1985.
- A maximum of 31 EFA sets for Spacelab missions will occur in 1985 and 1986 (delivery required every 12 calendar days).
- Experiment Flight Applications software sets will be classified as either new or reflight with the development activity for reflights varying, depending on number of reflights per payload. It should be noted that even though many experiments will reflly, they may fly with a different payload (experiment mix).

### 2.2.3 DISCUSSION

To determine the operational environment which must be supported by the EFA software development concept, a detailed analysis of the Shuttle Sortie Mission Model was performed. The purpose of this analysis was primarily to determine peak software development activity periods and to determine the delivery support requirements. These activity parameters are essential inputs in determining the development concepts and the support facility requirements for EFA software development.

The summary results of the Sortie Mission Model analysis are shown in Table 2.1. Subsequent paragraphs will address the appropriate contents of the table and will establish the rationale used in developing the detailed data.

REFERENCE "The October 1973 Space Shuttle Traffic Model"  
NASA Technical Memorandum NASA TMX-64761,  
Rev. 2, by Shuttle Utilization Planning Office, Program  
Development, January 1974, Tables 4 and 9

Table 2.1. Spacelab Traffic Model Summary (Experiment Flight Applications Impact)

N = NEW  
R = REFLIGHT  
NN = Non-NASA  
No. 1 = Number of Instructions

TYPE EXPERIMENT	YEAR TYPE MISSION	80		81		82		83		84		85		86		87		88		89		90		91		TOTALS			
		N	H	N	R	N	R	N	R	N	R	N	R	N	R	N	R	N	R	N	R	N	R	N	R	N	R	T	
ASTRONOMY STELLAR	No 1	0	0	1	35600	1	49840	2	24920	3	106800	2	113920	3	135280	2	85440	3	117480	3	35800	3	28480	3	21920	3	15	19	34
ASTRONOMY SOLAR PHYSICS	No 1	1	35600	1	14240	1	35600	2	24920	2	10680	1	53400	2	39160	2	14780	2	71200	3	39160	3	17800	2	14240	2	8	17	25
HIGH ENERGY	No 1	2	71200	2	71200	4	58960	4	42720	4	28480	4	14240	4	42720	3	21360	3	17800	3	14240	3	10680	3	10680	3	5	33	38
ATMOSPHERE PHYSICS	No 1	1	1	1	35600	1	14240	1	35600	1	60520	3	21360	4	28480	4	17800	4	17800	3	10680	4	14240	4	10680	3	3	27	30
LIFE SCIENCES	No 1	1	1	2	17800	2	7120	1	49840	2	17800	2	7120	2	7120	2	7120	2	10680	3	10680	3	10680	3	10680	3	2	26	28
SPACE TECH	No 1	2	2	2	99680	4	49840	4	35800	4	21360	4	14240	4	14240	4	14240	4	14240	4	14240	4	14240	4	14240	4	4	42	46
OFF OF APPLIC	No 1	2	2	2	28480	2	31360	2	14240	1	7120	1	7120	2	7120	2	7120	2	7120	2	7120	2	7120	2	7120	2	4	20	24
SPACE PROC	No 1	2	3	3	113920	13	74780	13	49840	13	48280	13	64080	13	64080	13	64080	13	64080	13	64080	13	64080	13	64080	13	3	133	136
TOTALS Missions	No. 1	9	1	8	10	2	27	2	28	5	28	4	32	5	30	4	29	5	30	34	34	34	34	34	34	44	317	361	
Instructions		334640	416520	319720	277680	277680	295040	295480	338200	281920	320400	195800	187320	15360															
NO NASA FLT APPLICATION PACKAGES	10	18	18	21	27	27	33	36	35	33	35	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	34	361	
NO NASA SPACELAB FLT SETS	9	18	18	21	27	27	31	31	31	26	30	29	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	305	
NO N/N SPACE LAB PACKAGES WITH NASA							4	6	6	4	1	4	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	28	
NO N/N SPACE LAB FLTS WITH NASA							1	3	3	2	1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	12	
NO N/N FLTS/ INTEG								2	2		2							2	2								7		
NON-SPACELAB FLTS FLOWN W/ SPACELAB PAY LOADS				4	5	10	7	9	9	3	7	7	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	72	
FOREIGN N/N FLT APPLICA- TION SETS	2	2	2	3	3	3	3	7	11	7	11	7	11	7	11	7	11	7	11	7	11	7	11	7	11	7	74		

\*INCLUDED IN "NASA SPACELAB FLIGHT SETS"

#### 2.2.3.1 Software Development Analysis

As illustrated in Table 2.1, each experiment is divided into new flights and reflights. To determine the magnitude of the software development activity, it was necessary to estimate the number of instructions to be generated for each classification.

##### New Flight Analysis

For estimating new flight requirements the Spacelab Sortie Payload Software Sizing Analysis report (Item 26-List of References) was utilized. This report considered a representative set of 13 Spacelab experiments and developed the number of instructions required to support major software functions of each. The results of this analysis are summarized in Table 2.2 and indicate the following:

- The average new flight application software package requires 8,000 instructions for monitor and control functions, 27,600 instructions for scientific data processing functions, and 20,000 instructions for the operating system.
- All flights require extensive computational support. (Note that LS-04-S has a dedicated processor.)
- The average payload contains nine sensors.
- One-half of the typical payloads requires computational support of less than 200 KOPS. The remaining payloads require processing which cannot be performed totally on board with current state-of-the-art computers.
- The memory requirements for EFA software can be supported by state-of-the-art computers.

##### Reflight Analysis

As illustrated in Table 2.1, the number of reflights to be supported far exceeds the number of new flights. For this reason, a detailed analysis was performed to ensure that the reflight impact was properly evaluated.

Analysis of the behavior of flight software development for past Saturn and Skylab flights indicated that a significant change activity continues after initial development of a software system. This change activity is attributable to the following factors:

- Improvement in software performance to better support experiment/mission objectives.

Table 2.2. Spacelab Payload Software Sizing Table

	Monitor & Control	Scientific Processing	Total	KOPS per Second	Sensors
1. AS-31-S	5,550	27,900	33,450	125	6
2. SO-01-S	8,750	29,750	38,500	38,167	18
3. HE-11-S	4,650	24,600	29,250	203	4
4. AP-05-S	8,700	28,350	37,050	117	7
5. OP-01-S	13,170	34,950	48,120	2,570	16
6. CN-01-S	8,400	30,700	39,100	1,201	8
7. SP-S	10,500	30,700	41,200	111	5
8. EO-05-S	4,600	20,650	25,250	2,832	3
9. LS-04-S	None	None	None	None	0 * See Note
10. LS-07-S	12,000	27,400	39,400	186	15
11. ST-01-S	8,250	32,150	40,400	13,491	18
12. ST-04-S	7,900	28,200	36,100	1,769	4
13. ST-08-S	3,150	15,900	19,050	37	5
Average		27,800	35,800	---	9

NOTE: LS-04-S contains a special purpose computer and, as such, imposes no requirement on the CDMS.

REFERENCE: Spacelab Sortie Payload Software Sizing Analysis, IBM No. 74W-00059, dated 27 February 1974.

A standard CDMS Computer Operating System of 20,000 instructions was assumed and is not included in the above numbers.

- Resolution of problem areas identified as a result of operational utilization.
- New or improved experiment hardware to be supported.

The characterization of the Spacelab enhancement activity shown in Figure 2-2 is based on past experience on similar software development projects. As may be seen, the first reflight will characteristically require approximately 40% modification from the new EFA, the second 30%, the third 20%, and all subsequent reflights approximately 10%.

Having established the characterization of software modification percentages throughout its life cycle, this data was used with the number of reflights in the Sortie Mission Model to establish the Spacelab reflight software impact. The approach used was to:

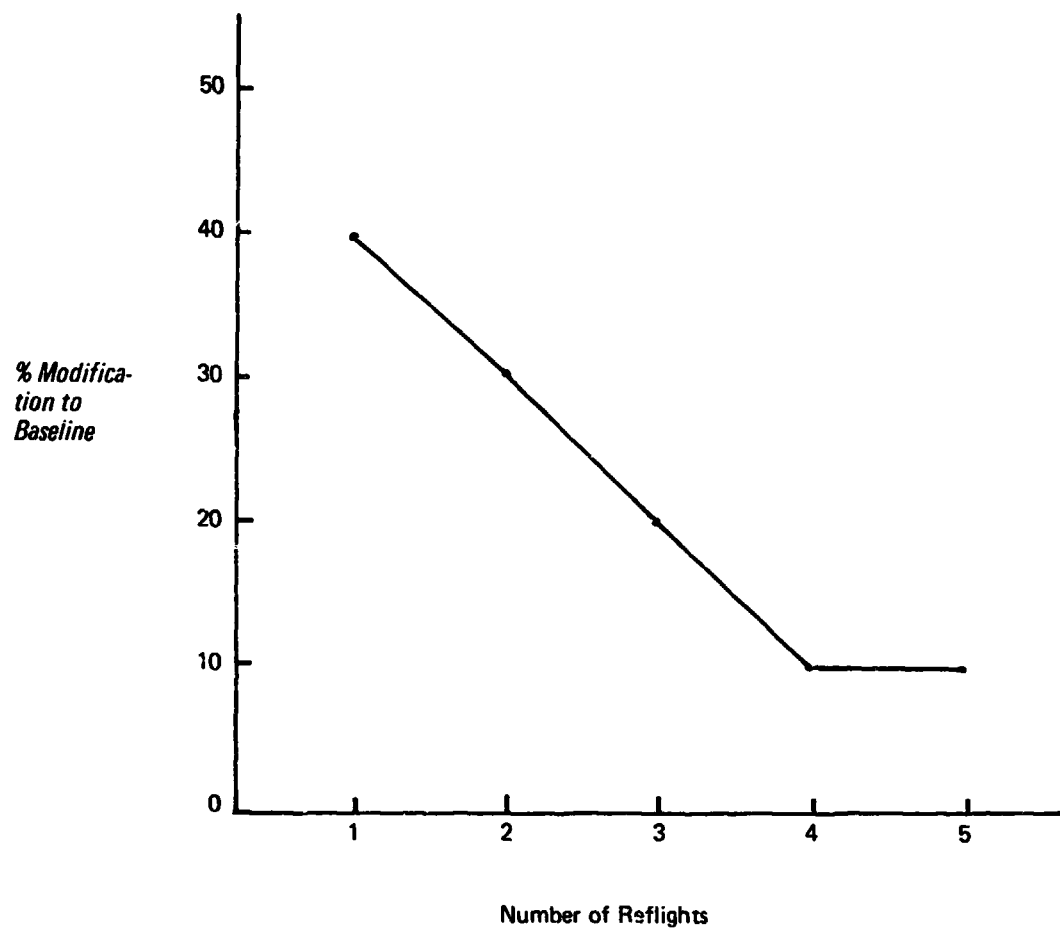
- Determine the number of reflights for each experiment code.
- Prorate the percentage change according to the number of the reflights.
- Apply these percentages to the average new flight instruction size (35,600) for each reflight.
- Total the resulting reflight development impact on a yearly basis.

The resulting totals, when combined with the new flight development activity, results in total development burden on a yearly basis. This data is summarized in the totals row of Table 2.1.

Knowing the total number of instructions to be developed is not sufficient input for establishing a development load factor. The number of software packages and sets in process at one time must also be understood. To arrive at this number, it was assumed the average application package manufacturing process would require 6 months; this would include flight application package integration/verification. As indicated in Table 2.1, there are more flight application packages than there are flight application sets. This results from combining the traffic model's payloads to generate flight configurations. As indicated, the maximum number of packages undergoing development is 36 in 1985. These packages are then combined into 31 flight application sets.

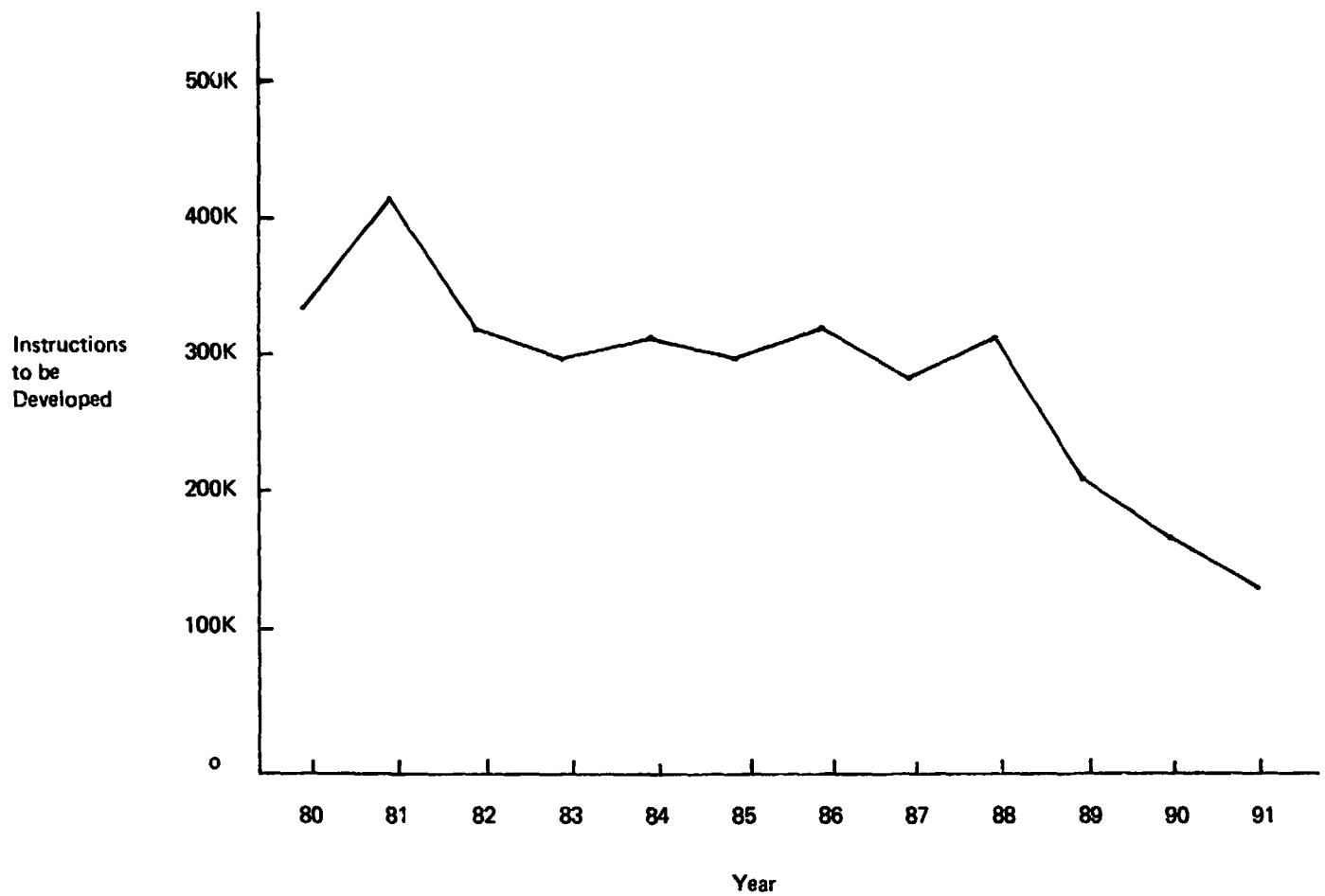
#### Development Analysis Summary

Combining the development requirements for both new flights and reflights resulted in the total development burden to be supported on a yearly basis. As may be seen in Figure 2-3, the peak activity year is 1981 with approximately 416,000 instructions to be developed. This indicates that the early peaking of activity will necessitate full development capability early in the program.



*Figure 2-2. Projected Experiment Flight Application Software Modification Activity*





*Figure 2-3. Instructions to be Developed/Year for Experiment Flight Application Packages*

### 2.2.3.2 Software Delivery Analysis

The capability to generate deliverable Experiment Flight Applications software sets with the necessary configuration management controls is an important factor to be considered in the Experiment Flight Applications development concept. The Sortie Mission Model was analyzed to determine the delivery requirements to be supported.

The results of this analysis are summarized in Table 2.1 on a yearly basis and are shown pictorially in Figure 2-4. As may be seen, the set delivery profile lags considerably behind the development activity because of the increasing number of reflights as the Spacelab program matures. The maximum number of deliveries reaches 31 in 1985 and 1986 and will require that a delivery be accomplished every 12 calendar days (365 days/year/31 deliveries/year).

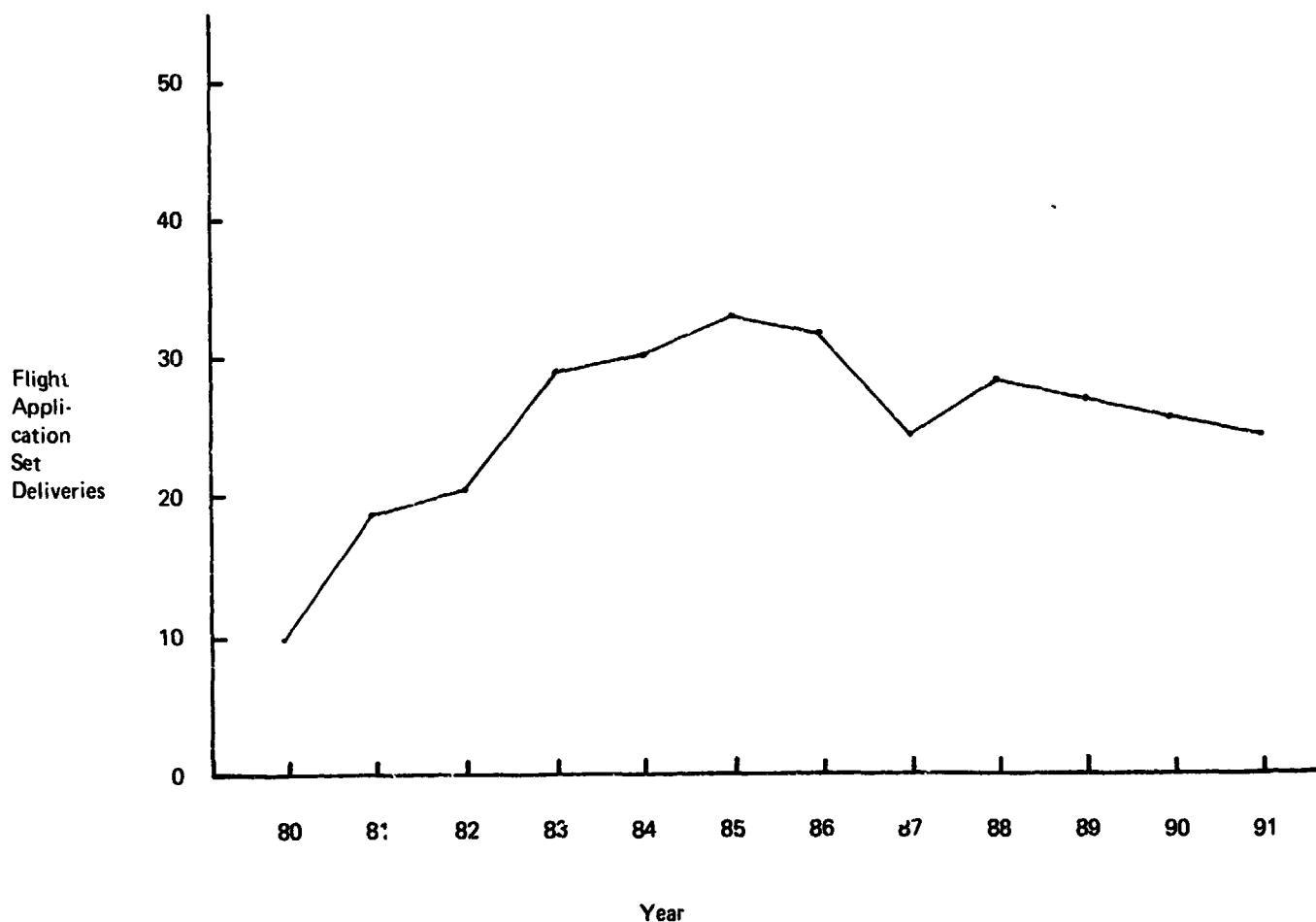


Figure 2-4. Spacelab Experiment Flight Application Delivery Profile

## 2.3 PI TO PI FLOW ASSESSMENT

### 2.3.1 THEME

Current Spacelab program plans call for each Principal Investigator (PI) to be responsible for his experiment (hardware and software). The PI involvement in experiment-associated software must be projected to understand the Experiment Flight Applications (EFA) software development environment. To define this involvement, the flow of the experiment from definition to post-flight analysis was analyzed.

### 2.3.2 CONCLUSIONS

Analyses of the PI to PI experiment flow have identified the following factors which influence the EFA development concept:

- The PI will be responsible for generation of the experiment-associated software requirements.
- To develop EFA software in house, the PI must be provided with a CDMS Simulator or CDMS Hardware.
- NASA will be responsible for integration of the experiment hardware and software.
- The PI will be responsible for development of post-flight experiment analysis software.

### 2.3.3 DESCRIPTION

The PI has been defined as that individual or organization technically responsible for a Spacelab experiment. This responsibility includes all phases of experiment development and utilization and encompasses both the hardware and software required in all phases. The definition of the PI's involvement in the software development process is essential in establishing an experiment software development concept. Analyses of available NASA plans and technical discussions with NASA study team counterparts have established the following data.

#### 2.3.3.1 Classes of PIs

A significant factor in the determination of the experiment software development concept will be the capability of the PI to support a software development activity. To establish an understanding of the range of support that must be provided the PI, the following classes of PIs have been established.

- PIs who want NASA to develop their experiment software. Examples are PIs who are NASA employees, PIs who have high interest in their field but little desire to program, or PIs with relatively minor software requirements.

- PIs who want to develop their experiment software on the NASA STIL. Examples are a PI from a college that has limited data processing facilities or a NASA/PI trained in programming.
- PIs who will develop their own software in-house using NASA supplied development facilities. Examples are non NASA PIs that are responsible for many payloads.

#### 2.3.3.2 PI to PI Flow

Through interface with MSFC counterparts, a conceptual definition of the PI's involvement in experiment flow was established. The conceptual overall flow is shown in Figure 2-5, and the envisioned PI's involvement in each activity is defined in the following paragraphs.

##### Experiment Definition

The initial activity accomplished in the experiment flow is the experiment definition. The PI will perform the definition activity and will include hardware and software requirements needed in support of the experiment. The definition of the experiment will be closely coordinated with NASA.

##### Experiment Selection

Upon completion of experiment definition, the PI will present the experiment to a NASA selection board responsible for selection of Spacelab experiments and allocation of experiments to particular Spacelab missions. The selection board will review the experiment along with alternative experiments and will provide the "GO/NO-GO" decision for inclusion of the experiment into the Spacelab Mission planning cycle. Acceptance and allocation of an experiment will initiate development of the hardware and software.

##### Experiment Build

The experiment build activity will include the development of both hardware and software needed to achieve the experiment's scientific objectives. The software required in this activity includes:

- Predelivery Experiment Test/Checkout Software (if required)
- Spacelab Ground Applications Software (EGSE)
- Flight Applications Software
- Payload Mission Operations Software
- Post-Flight Data Reduction Software
- Scientific Data Analysis Software

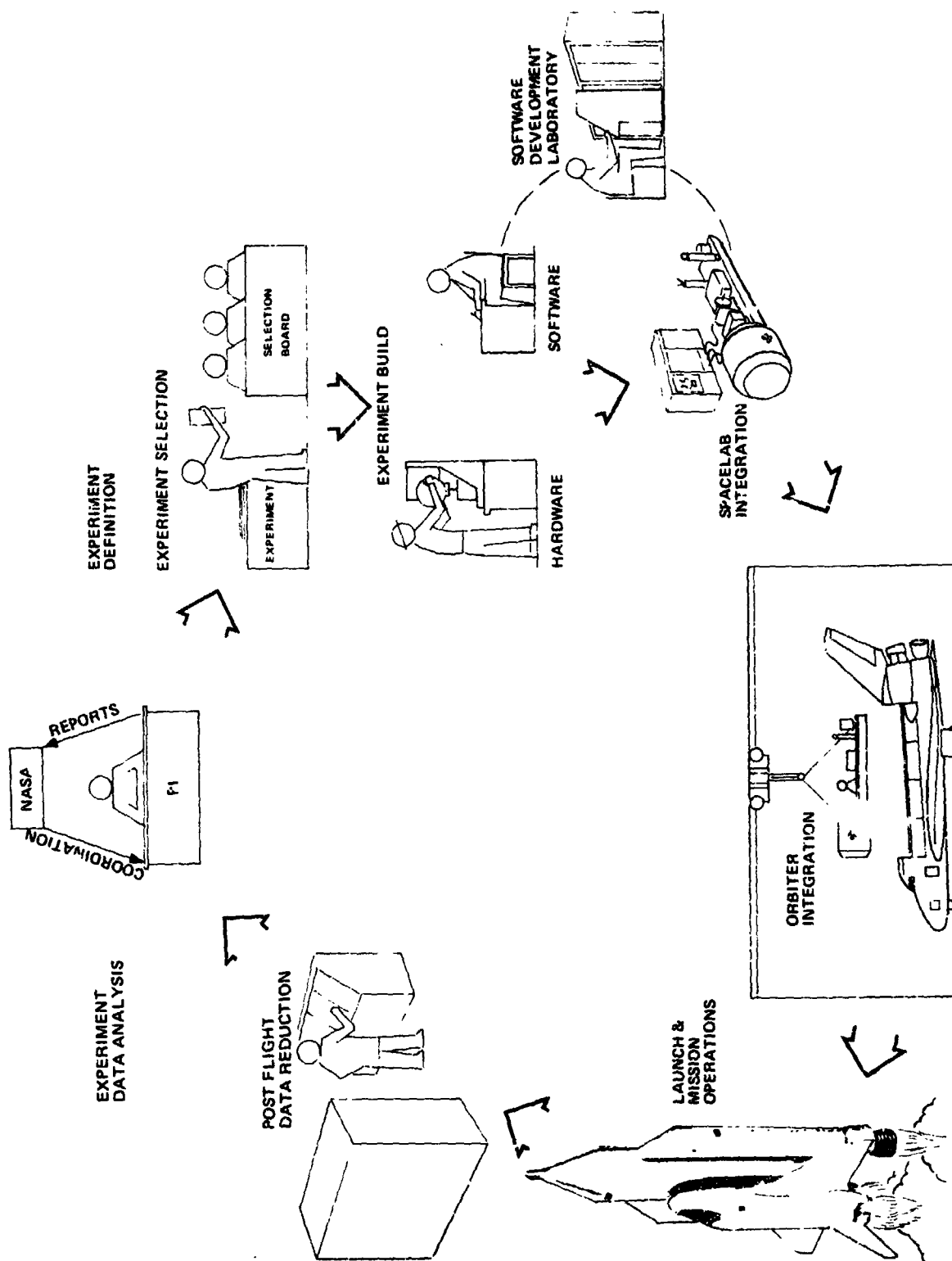


Figure 2-5. PI Flow

Each of the software items will be undergoing development in parallel with the experiment hardware and will require substantial software expertise on the part of the PI. The PI's participation in each of the areas is summarized in Table 2.3.

*Table 2.3. PI Participation in Experiment Software Development Tasks*

EXPERIMENT SOFTWARE	PI PARTICIPATION
Predelivery Experiment Test/Checkout	Responsible for development (software is totally experiment-dependent).
EGSE	Generates requirements.
Experiment Flight Applications (EFA)	Generates requirements - has option on whether to develop.
Mission Operations	Generates requirements.
Post Flight Data Reduction	Generates requirements.
Scientific Data Analysis	Responsible for development (software is totally experiment-dependent).

#### Spacelab Integration

Having completed the build process, the experiment will be integrated into the Spacelab. Interface testing will be performed to ensure system integrity. It should be noted that this point in the flow will be the first testing of the experiment hardware, common flight hardware and Experiment Flight Applications software as an entity. This integration/validation process will be a NASA responsibility.

#### Orbiter Integration

The Spacelab, containing the experiment hardware and software, will be integrated into the payload bay of the Space Shuttle Orbiter. Overall Orbiter/Spacelab interface tests will be performed prior to launch.

#### Launch and Mission Operations

During the on-orbit time of the Spacelab mission, the EFA software will control and monitor the operations of the experiment hardware. At the same time, mission operations software in the Payload Operations Center (POC) will gather realtime experiment data, provide the PI with the capability to monitor performance of his experiment, and provide PI/onboard system interaction capability.

### Post Flight Data Reduction

Prior to detailed analysis of experiment results, the data gathered in realtime and/or recorded onboard will be preprocessed by NASA. This process tests quality of data, converts the data to calibrated engineering units, and formats it for final processing by the PI. Because of the requirement to interface with the TDRSS and NASA ground network to perform this process, NASA will provide facilities and software to perform pre-processing.

### Experiment Data Analysis

The final step in the experiment flow will be the detailed analysis of experiment results by the PI on his facilities. The software used in this evaluation will be developed by the PI. Upon completion of detailed analysis, results will be distributed to the scientific community and any necessary requirements for reflight will be passed to NASA.

#### 2.3.3.3 PI Requirements on Experiment Development Concepts

In analyzing the total PI flow, the only PI activity which impacts the development concepts is the EFA software development. Primary emphasis must be placed on ensuring that options exist within EFA software development concepts to cover the widely varying software capabilities of the PIs. The options shown in Figure 2-6 represent the full range of PI involvement. As can be seen, the options provide the capability for the PI to develop the EFA software even though NASA is responsible for integrating the software into a payload flight configuration.

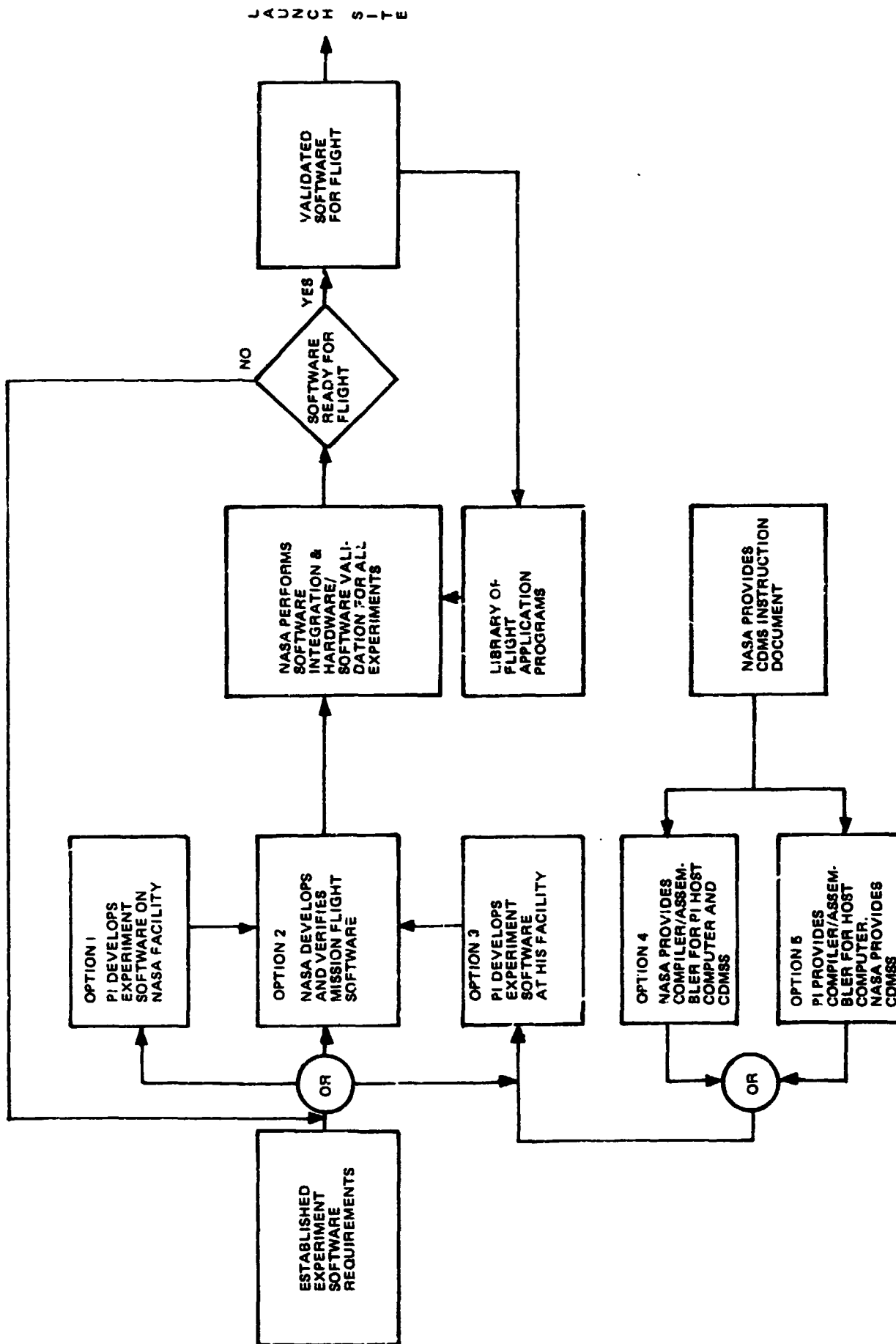


Figure 2-6. Experiment Software Development Options



## 2.4 PI/CDMS SOFTWARE INTERFACES

### 2.4.1 THEME

An analysis of the requirements for PI/CDMS software interfaces was performed to establish the impact upon the EFA software development concepts. This analysis determined requirements for real time PI interaction at three major operational sites, all of which affected the Spacelab Experiment Software Development Concepts (See Figure 2-7). Operational and programmatic considerations require that the PI or his representative (onboard crew, Payload Operation Center (POC) operator, and testing personnel) be provided the capability to communicate with the onboard experiment hardware for real-time control and evaluation. This communication interface must provide flexibility to meet the full range of nominal and contingency experiment operating and test modes.

### 2.4.2 CONCLUSIONS

Significant conclusions reached relating to PI/CDMS Software Interface are:

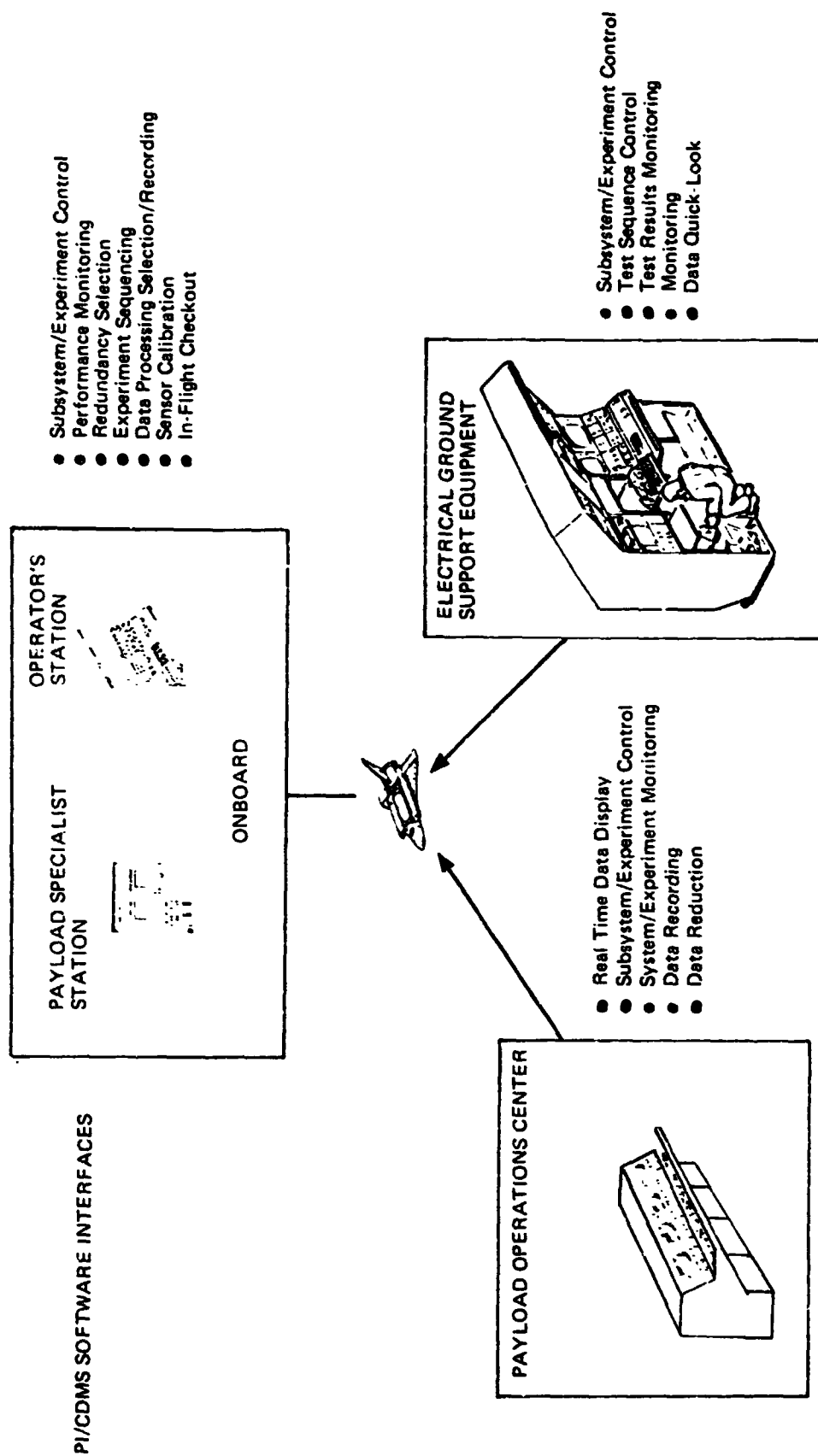
- Extensive and complex EFA Software will be required to support real-time experiment/PI interaction
- Special-purpose PI/Engineer oriented interface language will be required.
- Crew/Operator training will be required for each experiment/mission
- Experiment development concepts must provide for comprehensive definitions, design certification, and verification of Experiment/PI interactive capabilities

### 2.4.3 DISCUSSION

. analysis of the operational phases of experiment development identified the major PI/CDMS software interface areas. These areas are (1) the onboard CDMS Operator's Station or Payload Specialist Station, (2) the Payload Operation Center Experiment (POC) Operators Console, and (3) the Electrical Support Equipment Operators Console (EGSE). Space-lab system design requires that the CDMS provide all control interface to the EGSE and POC and that the EFA Software must share or perform all real-time PI interaction services during all experiment development phases.

#### 2.4.3.1 Interactive Control Requirements

Analysis indicates that interactive control of the experiment must be shared between the onboard operator and the POC operator, during flight, or the EGSE operator during integration checkout operations. Composite PI/CDMS interface requirements are summarized as follows:



*Figure 2-7. PI Interfaces will affect Experiment Software Development Concepts*

- Experiment Manual Operational Control
  - Power control
  - Mode control switching
  - Pointing or target acquisition
  - Sample rate modification
  - Experiment time line modification
  - Experiment schedule period modification
  - Inhibit/Enable automatic sequence commands
  - Selection of experiment redundant configurations
  - Individual sensor control (align, focus, etc.)
  - Dynamic scientific data routing
- Experiment Data Display and Control Processing
  - Real time engineering unit display of critical parameters
  - Tutorial display of experiment normal processing
  - Selected displays of experiment data related to data collection and storage
  - Experiment data quality parameters for realtime decision making
- Sensor Calibration/Checkout
  - Individual sensor operational checkout
  - Individual sensor calibration
  - Experiment pre-operation checkout
- Status/Performance Monitoring
  - Exception monitoring during automatic sequencing
  - Crew alert for abnormal conditions
  - Crew alert for significant experiment events
  - Limited individual sensor trend analysis
  - Provide CDMS experiment processing load data analysis

To meet these interaction requirements, onboard CDMS hardware will consist of:

- CRT display system providing for text and graphic data presentation
- Alpha-numeric keyboard providing for effective man/machine data entry
- Manual switches providing for hardline experiment control as well as CDMS function commands.
- Panel lights providing for CDMS and hardline crew discrete attention/status indicators

- Orbiter/Spacelab telemetry downlink capability to include CDMS Status as well as experiment data
- Orbiter/Spacelab telemetry command uplink capability to receive command from POC or EGSE

#### 2.4.3.2 Interactive Software Requirements

To meet interactive requirements EFA, EGSE and POC software must support the utilization of the Spacelab interactive hardware in the most cost effective manner. Due to limited control and display capability, software must provide for multi-function capability on non-hardline displays and switches. Interfaces must be standardized to prevent excessive crew/operator retraining between missions and to reduce possibility of manual errors. Software must provide self protection to prevent invalid interactive commands from resulting in invalid or destructive actions by the CDMS. Last but not least, is the requirement that all displays be presented in such a manner and format that the crew/operator will be able to quickly recognize conditions presented and respond as required.

#### 2.4.3.3 Interactive Language Requirements

During the analysis, it was established that a common interactive language with a PI/Engineer oriented syntax and PI/Operator display format is required to:

- Minimize input errors
- Minimize crew/operator training
- Reduce interface problems
- Maximize flexibility and utilization
- Enhance the cost effectiveness of operational experiment development

Whenever a hardware/software system is designed flexible enough to meet the requirements of the Spacelab experiment/PI interaction, it must also provide adequate assurance that the system will not fail as a result of invalid operator input. Therefore, PI/CDMS software interface requirements will significantly affect the development concepts in the areas of design definition, design certification, verification and validation.

## 2.5 EXPERIMENT CHECKOUT/VERIFICATION TECHNIQUES

### 2.5.1 THEME

The schedule for checkout and verification of the experiment hardware and software is a major driver in the establishment of facility requirements and overall experiment flow. The stringent timeline condition which will exist levies unique requirements on checkout/verification techniques.

### 2.5.2 CONCLUSIONS

- Due to the restrictive time frame for hardware/software interface validation and checkout, experiment hardware and software must be thoroughly tested prior to payload integration.
- NASA must provide a facility for software verification.

### 2.5.3 DISCUSSION

The Spacelab Ground Operations Plan (Item 1, List of References) established the overall flow which is depicted in Figure 2-8. As may be seen in the figure, the process of checkout and verification is initiated with the experiment hardware tests; and upon successful completion of these tests, the hardware/software validation and payload integration activities are performed. These activities culminate in the integration of the payload into the Spacelab and the Spacelab into the Shuttle.

#### 2.5.3.1 Premission Timeline

The following four pertinent premission time frames have been extracted from the Ground Operations Plan:

1. Payload integration function is allocated seven working days-- of which sixty-eight hours are allotted for interface validation and checkout at the Central Integration Site (CIS).
2. Flight Readiness Test is allocated 40 hours for final integrated system tests at the launch site.
3. Spacelab/Shuttle Orbiter verification is allocated two hours for interface verification.
4. Payload Final Services are allocated four hours.

From those time frames, it may be seen that the timeline at the launch site will be so severe that any error could abort the mission. Therefore, the payload must be fully verified before transporting to the launch site. Figure 2-9 represents the timeline for CIS payload

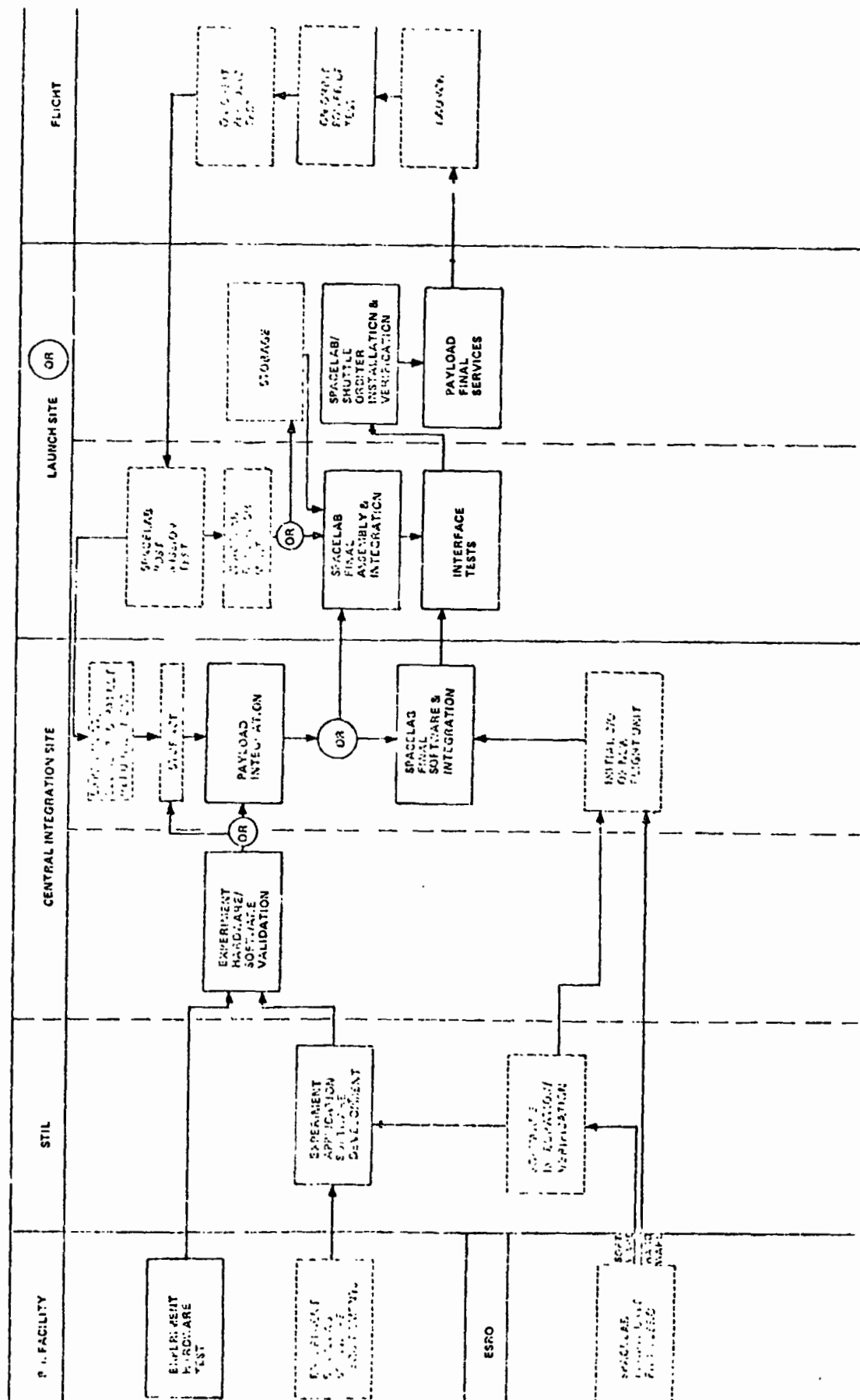


Figure 2-8. Overall Spacelab Flow with Emphasis on Experiment Checkout

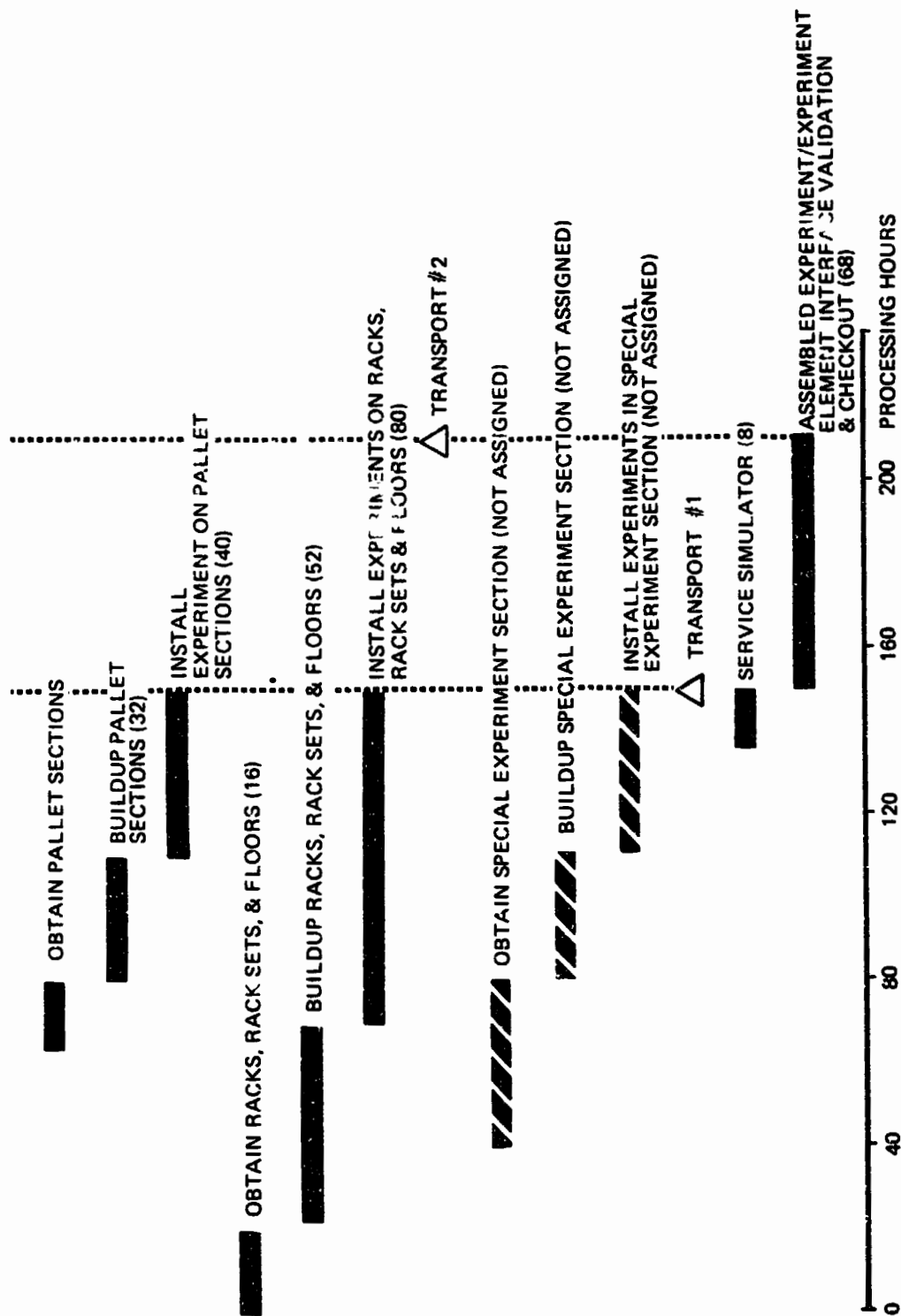


Figure 2-9. CIS Payload Integration Time Line

integration, and Figure 2-10 represents a projected expansion of the 68 hours allocated to Experiment/Experiment Element Interface Validation and Checkout.

As indicated in Figure 2-10, of the seven days allocated for payload integration, only sixty-eight hours are allotted for interface validation and checkout at the CIS. This 68-hour time frame will be a sensitive factor in the total Spacelab flow and will be especially critical when the payload integration involves new flight configurations.

#### 2.5.3.2 Hardware/Software Validation

Validation of experiment hardware and software may be separated and considered as two separate functions: validation of reflights and validation of new flights. Reflights imply the hardware has flown successfully on previous missions. For the reflights, the 68 hours of interface validation and checkout at the CIS should be sufficient since careful application of configuration control can minimize the probability of the existence of incompatibilities.

Validation of up to nine new flights per year poses a potential problem when each integration is constrained to only a 68-hour time span within the allotted seven day payload integration span. The 68 hour timeline now becomes a critical factor in the total Spacelab flow since this will be the first opportunity for validation of interfaces between Spacelab hardware and software. It must be apparent that the solution of any minor incompatibility existing between the hardware/software interfaces may require much longer than the 68 hours allotted. This situation has the potential of totally disrupting the schedule of events for the subject payload, and all other payloads within the operational flow, and is identified here as a major problem to be addressed.

#### 2.5.3.3 Software Testing Prior to CIS Delivery

A concept for the solution to the validation problem is that flight application software be thoroughly verified prior to integration, and strict configuration control be maintained to minimize incompatibilities at integration. Onboard experiment application software packages must be developed and verified in an environment that simulates the Spacelab as closely as possible. This requires a copy of the CDMS and a facility that simulates the Spacelab, the Orbiter, the Payload Operations Center and the EGSE.

The checkout technique of providing facilities that provide high fidelity simulation for software verification prior to integration, and a facility for hardware/software integration has proved successful in the Saturn, Apollo, and Skylab programs. The Shuttle Software Development Laboratory (SDL), currently being developed at Johnson Space Center, provides these necessary services for Shuttle onboard software development efforts. High quality software with few integration and operational problems has been the constant product of this technique.



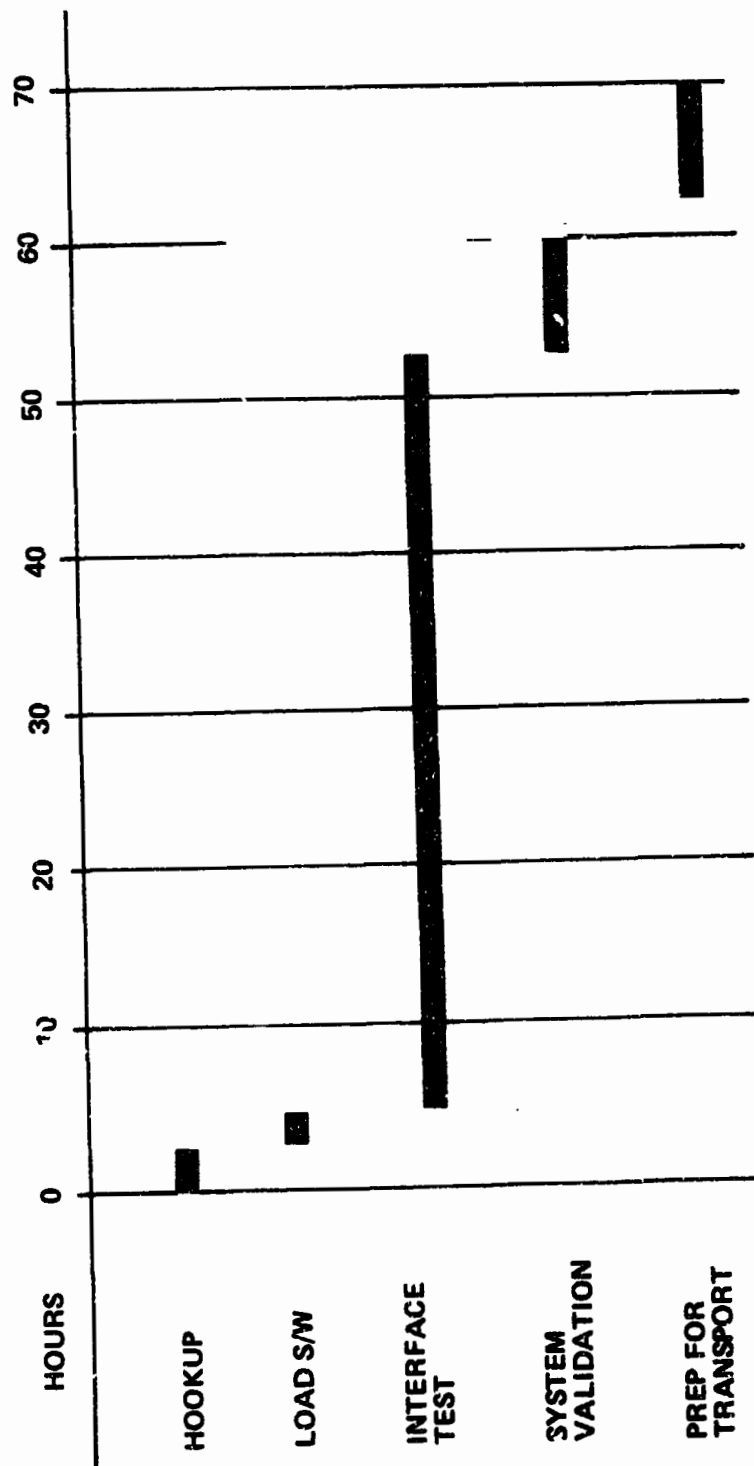


Figure 2-10. Experiment/Experiment Element Interface Validation and Checkout

#### 2.5.3.4 Experiment Testing Prior to CIS Delivery

Current plans call for each experiment to be independently developed by the PI. The experiment hardware will be fully checked out and validated on the PI's premises prior to delivery to the CIS. It is anticipated that all software activities required for experiment checkout prior to CIS delivery will be the responsibility of the PI with NASA involvement as necessary to ensure compatibility with mission integration requirements.

## 2.6 DEVELOPMENT PHILOSOPHY DETERMINATION

### 2.6.1 THEME

Definition of an Experiment Flight Applications (EFA) development philosophy, which satisfies the requirements established during analysis of the Spacelab Traffic Model, PI to PI Flow, and Experiment Check-out/Verification Techniques, is required to provide uniform inputs to other study tasks. The development philosophy selected is flexible in concept but rigid in control in order to meet all Spacelab program objectives and service all classes of PI involvement in EFA software development.

### 2.6.2 CONCLUSIONS

The major conclusions of this section are:

- Experiment Flight Applications software must be developed in a disciplined, controlled environment.
- Experiment Flight Applications software development concept characteristics are compatible with state-of-the-art software development concepts.
- Proposed development process is compatible with all development options established for EFA software.

### 2.6.3 DISCUSSION

In formulating the EFA software development philosophy, many things must be considered and weighed against experience and known processes which have proved effective. Figure 2-11 graphically represents the major considerations used in developing the three identified products of this task: Proposed Experiment Flight Applications Software Characteristics, Proposed Experiment Flight Applications Software Development Process, and Proposed Experiment Flight Applications Software Development Responsibilities. The following paragraphs will first address the development philosophy considerations relevant to the development concept for experiment applications software and then will discuss the characteristics of the selected development concept.

#### 2.6.3.1 Development Philosophy Considerations

The prime item of importance in establishment of the development philosophy was a broad functional review of the Experiment Flight Applications Software requirements. As may be seen in Table 2.4, there is a high probability that major commonality will exist across the span of application requirements. It may be concluded from this that the basic philosophy must include a highly modular structure to take advantage of this commonality.

**MAJOR SOFTWARE DEVELOPMENT  
PHILOSOPHY CONSIDERATIONS**

- FLIGHT APPLICATION SOFTWARE REQUIREMENTS
- RELIABILITY REQUIREMENTS
- DEVELOPMENT TIMELINE
- TEST/INTEGRATION FACILITIES
- RELATIVE DEVELOPMENT COSTS

STUDY  
TEAM  
WORKING  
SESSIONS

**RESULTING FLIGHT APPLICATIONS  
SOFTWARE DEVELOPMENT PHILOSOPHY**

- PROPOSED FLIGHT APPLICATION SOFTWARE CHARACTERISTICS
- PROPOSED FLIGHT APPLICATION SOFTWARE DEVELOPMENT PROCESS
- PROPOSED FLIGHT APPLICATION SOFTWARE DEVELOPMENT RESPONSIBILITIES

*Figure 2-11. Development Philosophy Determination*

Table 2.4. Software Functions Required by a Representative Group of Experiments

FUNCTION	EXPERIMENT													
	AS-31-S	SO-01-S	HE-11-S	AP-05-S	OP-01-S	CN-01-S	SP-S	EO-5-S	LS-04-S	LS-07-S	ST-01-S	ST-04-S	ST-08-S	
DATA COMPRESSION	x							x		x				
IMAGE SIGNATURE ANALYSIS								x						
OTHER SIGNATURE ANALYSIS														
IMAGE PROCESSING														
BRIEFING MATERIAL CONTROL														
SCIENTIFIC DATA CONTROL														
GRAPHICS PROCESSING														
FOURIEH ANALYSIS														
AUTO CORRELATION														

It has been determined that Reliability Requirements of Experiment Flight Applications software vary between highly critical to low, depending upon the function of the experiment and the opportunity for reflight. The development philosophy encompassed with full spectrum of quality requirements. It must be noted here that, although there are lower performance parameters related to some experiment applications, there is an extremely critical requirement that Experiment Flight Applications software, if it fails, must fail safe.

The software Development Timeline and the Test/Integration Facilities were of major importance in determining the basic philosophy. Figures 2-12, 2-13, and 2-14 are presentation material developed and presented to NASA during the timeline analysis. Conclusions of timeline and facility analyses which impact the development concept were:

- Flight sets will be integrated by NASA.
- Software and hardware would first meet at the Central Integration Site.
- Timeline is critical from integration through launch.
- Software integration and verification must be performed prior to hardware/software integration and validation.
- Software development options which must be supported by the development concept are numerous.
- Many concurrent software development activities must be supported.

The development constraints established from timeline and facility analyses provide a new challenge to integrating, verifying, and validating flight software in a very short period of time. The critical period of time from integration to launch imposes a testing environment in which software/hardware integration/validation time is almost nonexistent. The philosophy to support this environment must provide quick, effective software integration, verification and validation; and, in addition, must provide the capability of strict configuration control at the module, package, and set levels of Experiment Flight Applications software development.

During the analysis of PI involvement and the assessment of software development sites other than the STIL, it was determined that development tools could become a major cost driver of the Spacelab program. Table 2.5 represents the five software development options which were considered. When selecting the two most desirable options (Option 1: PI develops on NASA STIL, and Option 2: NASA team develops software), Relative Development Cost was the driving factor. Even though Options 1

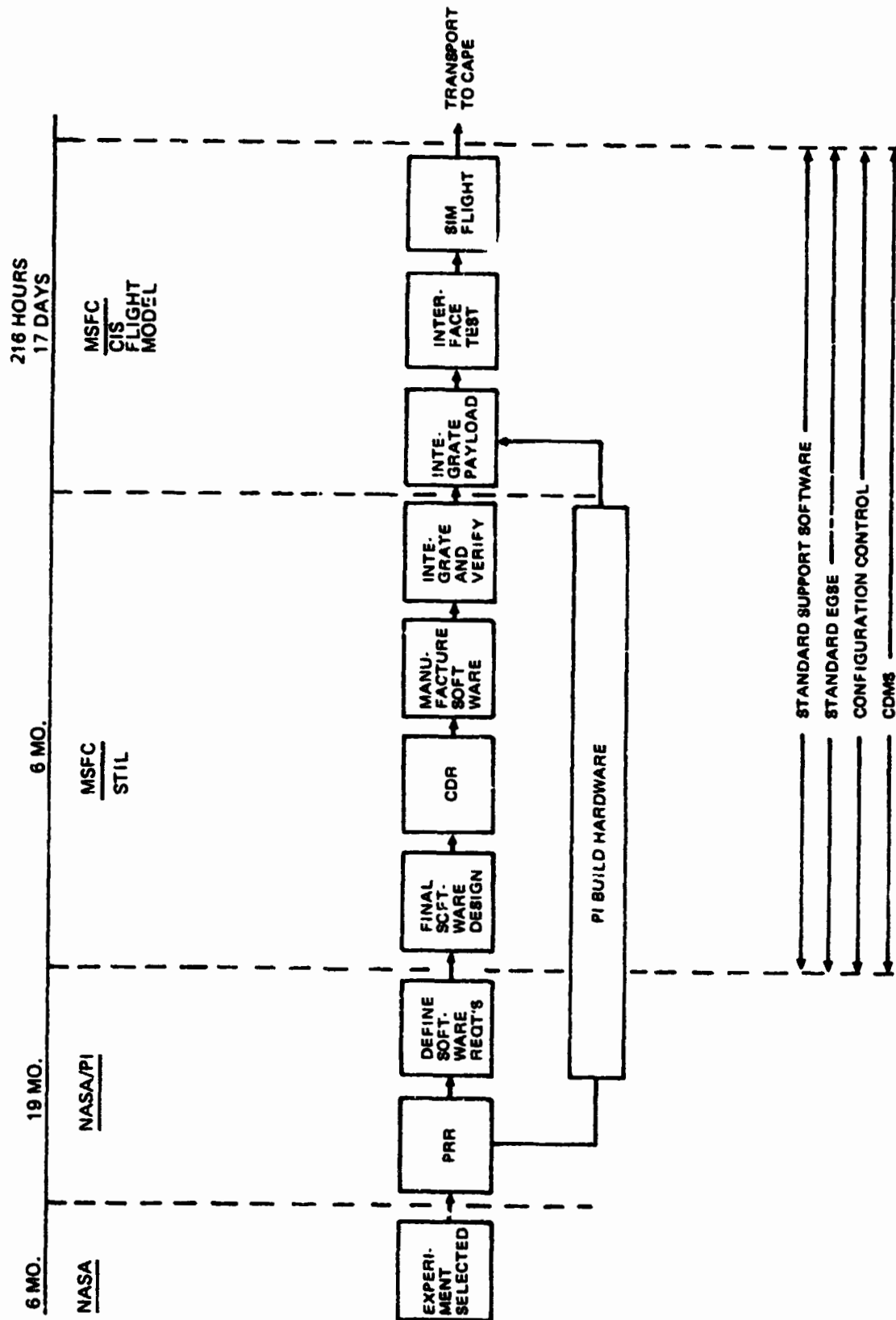


Figure 2-12. MSFC Software Development Timeline

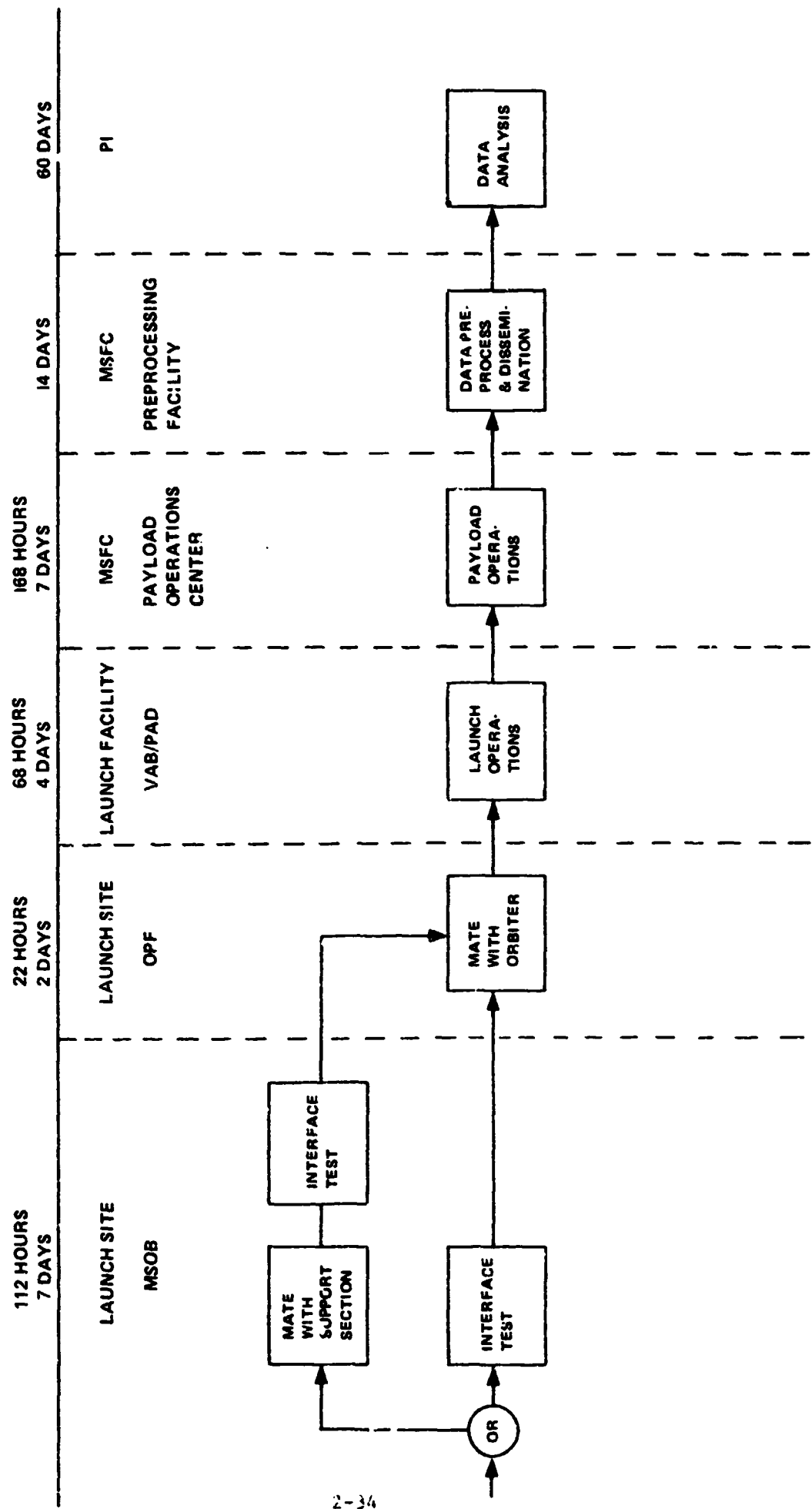


Figure 2-13. Launch and Prelaunch Timeline



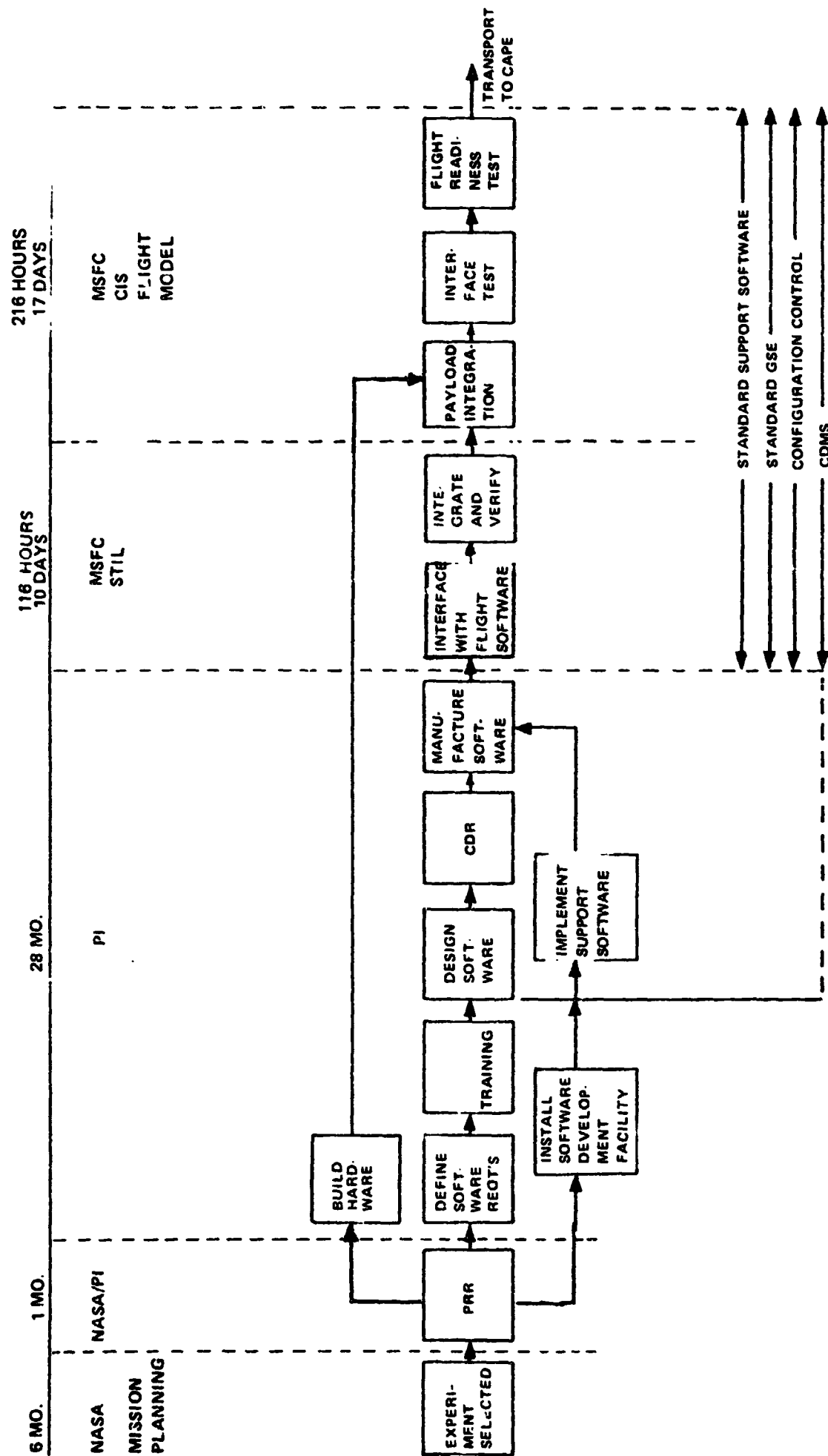


Figure 2-14. PI Software Development Timeline at his Facility

Table 2.5. Software Development Options

SOFTWARE DEVELOPMENT OPTIONS	LOW HOST COMPUTER COSTS	MINIMUM CDMS COSTS	MINIMUM SUPPORT SOFTWARE	MINIMUM SOFTWARE	HOL SUPPORT	LEAST SOFTWARE PERSONNEL TRAINING	FULL INTEGRATION TEAM VISIBILITY AND CONTROL
1. PI Develops on NASA STIL	YES	YES	YES	YES	YES	NO	YES
*a. Using NASA Defined Languages	YES	YES	YES	YES	YES	NO	YES
b. Using his Language	YES	YES	YES	YES	YES	YES	YES
*2. NASA Team Develops Experiment Software	YES	YES	YES	YES	YES	YES	YES
3. PI Develops on his NASA Supplied STIL	NO	NO	YES	YES	YES	NO	NO
4. PI Develops on his Facility using STIL Compatible Computer	NO	NO	YES	YES	YES	NO	NO
a. Using NASA Defined Language	NO	NO	YES	YES	YES	YES	NO
b. Using his Language	NO	NO	YES	YES	YES	YES	NO
5. PI Develops on his Facility using his Computer	NO	NO	NO	NO	NO	NO	NO
a. Using NASA Defined Language	NO	NO	NO	NO	NO	NO	NO
b. Using his Language	NO	NO	NO	NO	NO	YES	NO

\*Most Desirable Options

and 2 are the most cost effective, it is required that, in some cases, the development philosophy provide the capability for the PI to develop Experiment Flight Applications software offsite and integrate with STIL-developed software to form flight configured software sets.

#### 2.6.3.2 Experiment Flight Applications Software Characteristics

The analysis conducted within this task confirmed that a standard executive approach to the Experiment Flight Applications software set construction, customized to meet the unique environment of Spacelab, is the most cost effective and desirable method. Figure 2-15 proposes a structure which meets or exceeds all identified requirements. Explicitly, this structure provides for the key elements of independent experiment application software and common operating systems.

Independent experiment application software package development is characterized by low cost, responsive development keyed to the changing requirements of the PI/experiment hardware on each Spacelab flight. The common operating system, however, is characterized by stable non-changing software elements which comprise the majority of the onboard critical software. Figure 2-16 summarizes the key points of the proposed structure. The structure is fully compatible with and supports the proposed Experiment Flight Applications Software Development Process and Development Responsibilities.

#### 2.6.3.3 Experiment Flight Applications Software Development Process

The proposed development process, which is represented in Figure 2-17, meets all known requirements of the development philosophy and provides the flexibility necessary to implement new requirements as they are identified. A scenario of the key elements of the process is contained in the following paragraphs. This scenario covers only the software flow and does not consider the level or number of review points.

##### PI's Software Designer's Document

A key element of the total philosophy and development process is a composite set of well defined and documented Experiment Flight Applications Software Standards, Procedures, and Interface Definitions. These are established within the PI's Software Designer's Document and are a total set of "Rules" which must be followed in development of Experiment Flight Applications software. These "Rules" must be used by non-STIL as well as STIL developers to ensure a cost-effective compatible integration.

##### Experiment Definition

The experiment definition, requirements, and performance specification task is always the responsibility of the PI working with the software developer. This definition includes critical timing, processing, onboard testing,

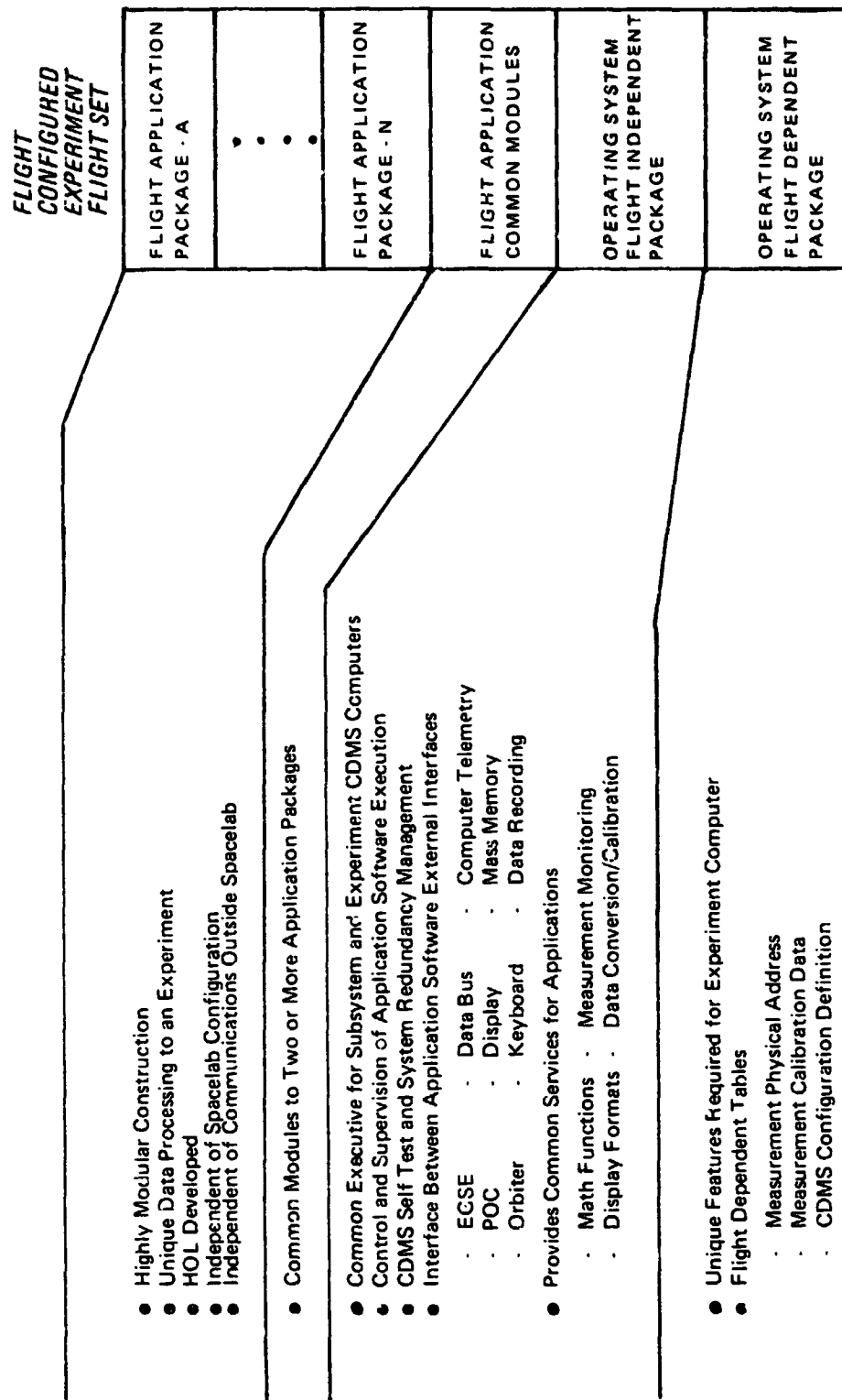


Figure 2-15. Proposed Flight Application Software Set Characteristics

**Independent Application Software Packages**

- High Modularity Environment
- Verification Level Variable on Criticality
- Low Probability of Application/Application Interference (Fail Safe)
- Configuration Control at a Package and Module Level
- PI Developed Software can Easily be Integrated at STIL

**Common Operating System**

- Cost Effective Subsystem/Experiment Computer Commonality
- All Major Stabilized Elements of Flight Configured Software Set
- Low Recurring Verification/Validation Requirements

*Figure 2-16. Key Elements of Flight Application Software Characteristics*

and performance definition of the Experiment Flight Applications package. A detailed review of the definition is conducted to ensure that the CDMS can meet the requirements being levied by the PI. Following agreement, the requirements and performance specifications are baselined and the Experiment Flight Applications software package design begins. As represented in Figure 2-17, up to 36 application package developments can be in process simultaneously.

#### Application Software Package Design

Using the baselined experiment definition, requirements, and performance specifications and functioning within the established standards, procedures, and interface definitions, the programming team begins the application software package design. Using the full principles of composite design concepts, the team breaks down the application into small manageable modules. The team makes full utilization of already developed modules in the module library and perhaps a package from the Flight Application Package Library. Detailed requirements and performance specifications on each module are established by the programming team. A design certification is performed on the application design to ensure that the design will meet the performance specifications. This may involve generation of development models to ensure that the design concepts and module structure will meet the requirements prior to committing to detailed module design. Following the design certification, the development is initiated with a top-down design implementation of the new application modules and verification plans are begun.

#### New Module Detailed Design

The development programmer performs a detailed design of the module while the backup programmer begins development of the verification procedures. It should be noted that several modules can be in production at one time for a package, using the top-down implementation process, and will require time phasing to ensure that all will be completed when required. A review is conducted following detailed design to assure that requirements are still being met or exceeded prior to committing the module to code and test.

#### New Module Code and Test

The development programmer codes the module in a selected HOL or machine language depending upon performance requirements. HOL selection would always receive positive consideration due to verification and compatibility considerations. Following the coding process, the module will be tested by the programmer. The module is then released to the backup programmer for verification.

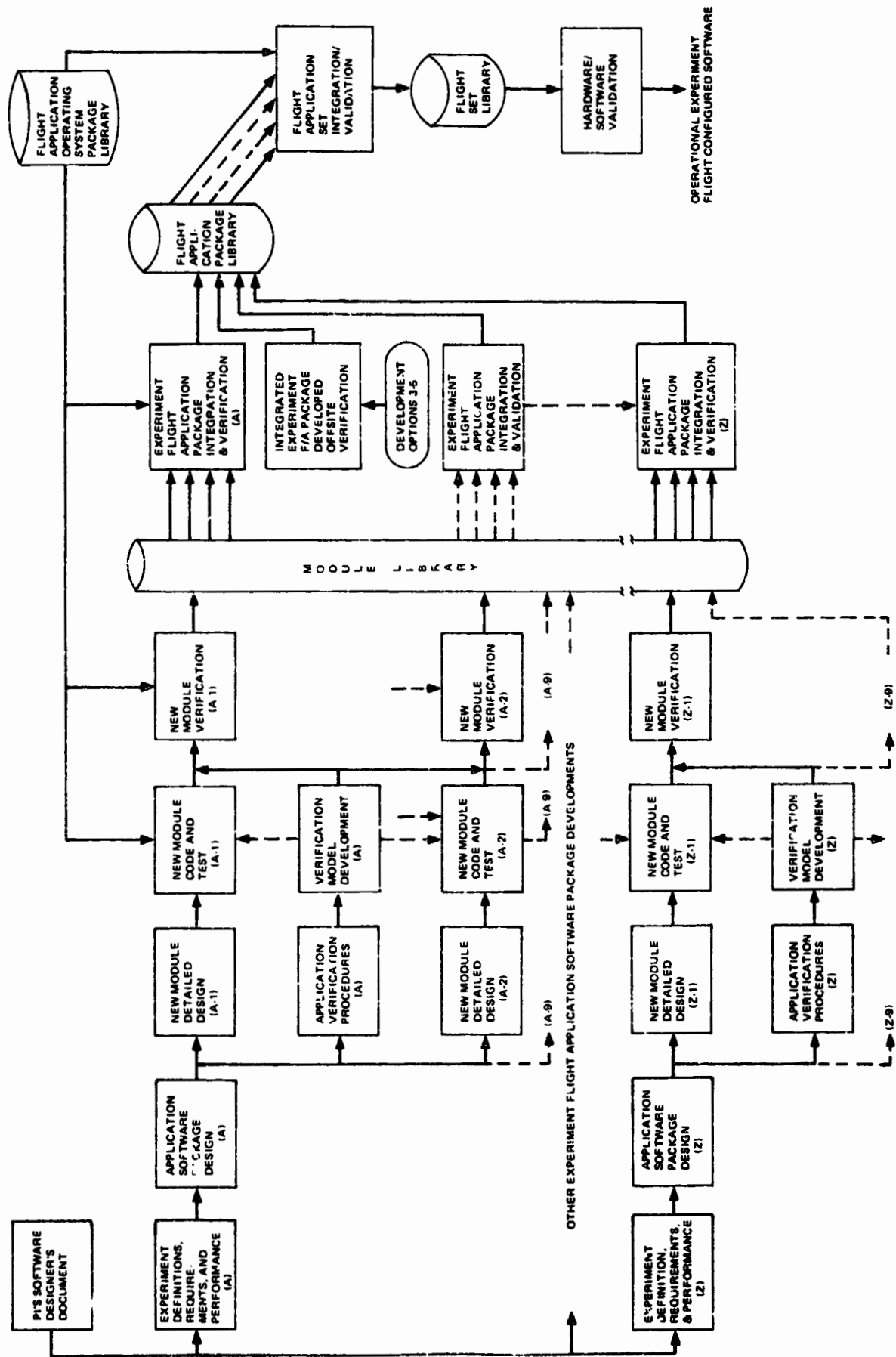


Figure 2-17. Proposed Experiment Flight Applications Software Development Process

### New Module Verification

The verification is accomplished on the module level to ensure that the module meets the requirements and performance specifications. Test cases and environmental models will be developed to test the module in a realistic system environment. These models are then placed under configuration control and later used in package and set verification. Any anomalies result in the program being returned to the code and test activity. After successful verification, the module is put into the module library under strict configuration control. The module is now ready for use in top-down testing and module integration for package verification purposes.

### Application Package Integration/Verification

When the last module developed is put into the module library, one final application overall integration/verification test is performed to ensure that the Experiment Flight Applications software package meets the requirements and performance specifications of the PI. The software package is now fully developed and verified as an Experiment Flight Applications package using high fidelity simulators as the test tool. If an anomaly is detected, the failing module is returned to the new module code and test or application software package design activity as appropriate. After full acceptance, the package is placed into the Experiment Flight Applications package library under configuration control. The environment models are also placed in the library with the same level of control for future use in set integration/verification.

### Flight Application Set Integration/Verification

Following the final selection of hardware for flight, the appropriate Experiment Flight Applications packages and operating system packages are linked together to form the experiment flight set. This integrated system is verified to prove interfaces and performance as a flight system using high fidelity system simulators. If an anomaly is detected, the set is deleted or corrected depending on the existing timeline. The flight software and simulators are now placed into the controlled Flight Set Library from which delivery to the CIS is made.

### Hardware/Software Validation

Within the CIS, the total system is exercised to ensure compatibility of interface. Due to the previous verification on high fidelity simulators and utilization of the stable common operating system, any anomalies detected in this phase will normally be the result of hardware failures or improper specification by the PI or differences between model and the flight hardware.



#### 2.6.3.4 Software Development Responsibilities

In all development processes, specific responsibilities must be defined to ensure compatibility. Table 2.6 proposes high level responsibilities related to tasks identified in the proposed development process. These responsibilities are related to the five options of software development identified earlier in the study. Depending on the development option, NASA/STIL responsibilities and configuration control will be established at different points in the development process. The following paragraphs will discuss the relationships between the development options and the development concept.

##### Option 1 (PI Develops on NASA STIL)

Using this option the software development plan is as depicted in Figure 2-17 with the PI being fully responsible for EFA definition, requirements, and performance specifications, application software package design, new module design, and new module code and test activities.

The NASA STIL team will establish the EFA verification procedures and will be responsible for developing the required models to be used in verification of the modules and application packages. The PI will function as a consultant in this activity. As each module is completed by the PI, it will be verified by the STIL team and the PI prior to being placed into the module library.

Following the development of the final module, the EFA package will be generated from existing and new modules. The integration and verification of this package is jointly done by the NASA/STIL team and PI through the use of STIL developed environment simulators. The package will then be placed into the flight application package library.

Once entry is made into the module and package libraries, configuration control is maintained by the NASA STIL team. The integration and verification of the flight set and the hardware/software validation of the flight set will be a NASA responsibility.

##### Option 2 (NASA Team Develops Experiment Software)

The PI is fully responsible for the EFA definition, requirements and performance specifications as he is in all options. Following this definition, the development responsibility is assumed by the NASA/STIL team. The flow shown in Figure 2-17 is followed exactly within this option. The details of this option are discussed in Paragraph 2.6.3.3.

##### Option 3-5 (PI Develops on Off-Site Facilities)

The off-site development options will allow the PI to build the software package; however, the NASA STIL team remains responsible for integration and verification of the flight set and hardware/software validation.

Table 2.6. Proposed Software Development Responsibilities

FLIGHT APPLICATION MAJOR DEVELOPMENT RESPONSIBILITIES	OPTION 1 - PI DEVELOPS ON NASA STIL			OPTION 2 - NASA TEAM DEVELOPS EXPERIMENT SOFTWARE		OPTIONS 3-5 - PI DEVELOPS ON OFFSITE FACILITIES	
	PI	STIL	STIL/PI	PI	STIL	PI	STIL
Experiment Definition of Requirements and Performance							
Application Package Design							
Module Design	PI			STIL		PI	
Module Code and Test	PI			STIL		PI	
Module Verification	STIL/PI			STIL		PI	
Module Library-Configuration Control	STIL			STIL		PI	
Application Package Integration/Verification	STIL/PI			STIL		PI	
Application Package Library Configuration Control	STIL			STIL		STIL	
Flight Application Set Integration/Verification	STIL			STIL		STIL	
Flight Set Library Configuration Control	STIL			STIL		STIL	
Hardware/Software Validation	STIL			STIL		STIL	

Operating within these options, the PI is responsible for total development of the EFA package. The integrated flight application package will be verified by the NASA/STIL team to ensure that there are no interface problems prior to placing it into the EFA package library. The STIL team will perform the necessary package and set verification testing. Within these options, no configuration control over the modules comprising the application will be provided by the STIL. Configuration control will exist at the package level only.

## 2.7 EXPERIMENT SOFTWARE REQUIREMENTS

### 2.7.1 THEME

As a result of the recommended development concept, the Experiment Flight Applications (EFA) software must adhere to certain design and development standards to ensure that development goals are achieved.

### 2.7.2 CONCLUSIONS

In support of development concepts, the following conclusions applicable to the Experiment Flight Applications software have been established.

- The software design must ensure separation of operating systems and application programs.
- The operating system software must support an interface language for PI interface and must protect the system from the user.
- The application software will support experiment-unique software requirements.
- A standard high-order language will be used for EFA software implementation.
- Adherence to development standards will be required.

### 2.7.3 DISCUSSION

Within the development concepts, the Experiment Flight Applications software must be organized and designed to allow ease of modification, rapid integration, and testability. The development concepts impact the Experiment Flight Applications software in the following areas:

- Design Requirements
- Language Requirements
- Operating System
- Applications System

The requirements levied on these areas are discussed in the following paragraphs.

#### 2.7.3.1 Design Requirements

The recommended development concepts for the Experiment Flight Applications software will allow the PI to develop the experiment-dependent software package, but the integrator/verification of the experiment

software into a flight set will remain a NASA responsibility. This separation of responsibilities will require design of the Experiment Flight Applications software such that the applications software (developed for or by PI) is independent of the operating system. In addition, the operating system/applications interfaces must remain standard across the Spacelab program to ensure that applications software can be reused for subsequent missions without modifications. To satisfy this capability, a modular design of the onboard software is mandatory, and rigid standards must be utilized to ensure modularity. The design approach is shown in Figure 2-18.

For system-build of the experiment software, the operating system design must allow the software system architect to specify, through tables, the characteristics of application subsystem to be supported for the payload. The types of characteristics to be provided include:

- Execution rates
- Execution times
- Priority of execution
- Size
- Location (main memory or mass memory)
- Module identification
- Input/output

Because of the varying configurations of Spacelab subsystem hardware, the system architect must also be allowed to specify symbolic representation of physical parameters to obtain hardware/software compatibility. This capability will allow automated reconfiguration of the physical environment without costly reprogramming and verification of Experiment Flight Applications software. These parameters will include the following types:

- Limits
- Measurement lists
- Physical addresses
- Test parameters

The man/machine interface capability required of the EFA software concept will require that the design allow the operating system to reconfigure application subsystems in realtime through PI requests. This request can come from the onboard consoles, from the Payload Operating Center, or from the EGSE.

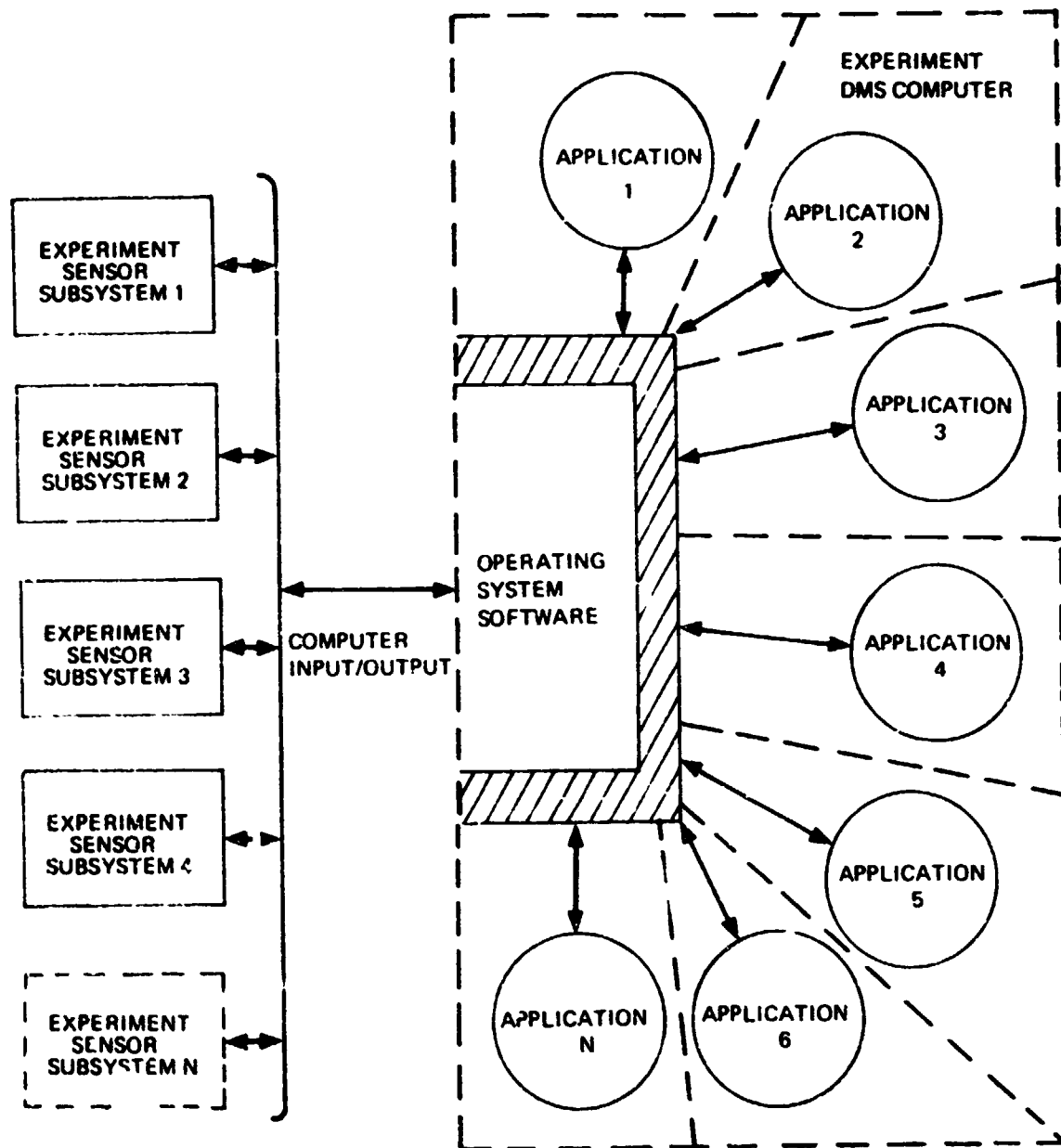


Figure 2-18. Experiment Flight Application Design Approach

### 2.7.3.2 Language Requirements

In support of the development concepts, the Experiment Flight Applications software language requirements can be divided into two types--Control and Display, and Development Languages.

#### Control and Display Language

The man/machine interface requirements will require that the operating system support a control and display language. Use of this language will provide flexibility to the PI in monitoring and controlling his experiment. Restrictions must be placed on the PI during on-orbit use of the language since incorrect utilization could potentially cause loss of an experiment.

#### Development Language

Because of the following advantages, a high order language will be required for Experiment Flight Applications software development:

- Can be effectively used by PIs who are not skilled programmers
- Ease of changes to software in short development cycle
- Increased programmer productivity to lessen manpower requirements
- Ease of software verification
- Ease of software integration
- Standardization of development languages across all PIs

The use of a high order language for onboard software may have disadvantages in increased memory utilization and execution times; however, if sufficient computer capacity is provided, the advantages will exceed the disadvantages.

### 2.7.3.3 Operating System Requirements

To support the development concepts (Section 2-6), the operating system for the onboard software must be a standard package throughout the Spacelab program. The operating system must provide the following capabilities:

- Protect the system operation from user error.
- Support control and display language.
- Control of realtime operation of the application packages.

- Standardized input/output with external environment.
- Provide fault tolerant operation.

#### 2.7.3.4 Experiment Application Packages

The applications packages will include that software which is experiment dependent. The applications packages will vary from mission to mission as the payloads vary. Because of the various payloads to be supported, the applications packages will contain independent modules which support such requirements as:

- Tracking/pointing control
- Control and display support
- Data acquisition and distribution
- Control/monitor of experiment engineering data
- Experiment control

Because of the many PIs to be supported, the application software development must adhere to development standards to ensure that the integration process can be rapidly and easily performed. Integration problems must be avoided or software will become the gating item in meeting launch schedules.



## 2.8 DEVELOPMENT TOOLS REQUIREMENTS

### 2.8.1 THEME

In support of the recommended Experiment Flight Applications (EFA) software development concept, development tools and supporting facilities are required in order to satisfy operational constraints.

### 2.8.2 CONCLUSIONS

A dedicated facility to be used for onboard experiment software development, test, and integration is required. This facility, known as the STIL, must provide state-of-the-art tools for use by programmers in the software development process.

### 2.8.3 DISCUSSION

Software development will require both tools and facilities. The requirements for both of these are discussed in the following paragraphs.

#### 2.8.3.1 Development Tools

With the rapid turnaround requirements and the magnitude of software development activity to be supported, development tools must be provided for the programmer to assist him in meeting the software development requirements. The tools identified in this study consist of the normal tools provided by a host computer center and the tools which are Spacelab unique. These development tools are summarized in Table 2.7, and the use of these tools within the development cycle is shown in Figure 2-19. The unique tools are briefly discussed in the following paragraphs.

#### Environment Models

The environment model must simulate the environment in which the onboard experiment software must function. The following models are required:

- Spacelab subsystems
- Shuttle Orbiter interface
- Experiments
- Payload Operations Center interface
- EGSE interface

These models are combined as required to form a realistic digital simulated environment for all phases of Experiment Application Software Development.

**Table 2.7. Development Tools Required for Spacelab Onboard Software**

SPACELAB ONBOARD SOFTWARE DEVELOPMENT TOOLS	SPACELAB UNIQUE
Environment Models	Yes
Development Models	Yes
CDMS Interpretive Simulator	Yes
Functional Simulator	Yes
Experiment Application High-Order Language	Yes
Experiment Simulation Language	Yes
Realtime Interactive Tools	Yes
On-Line Interactive Tools	No
Standard Assemblers/Compilers	No
Standard Utilities	No
Automated Configuration Management System	Yes

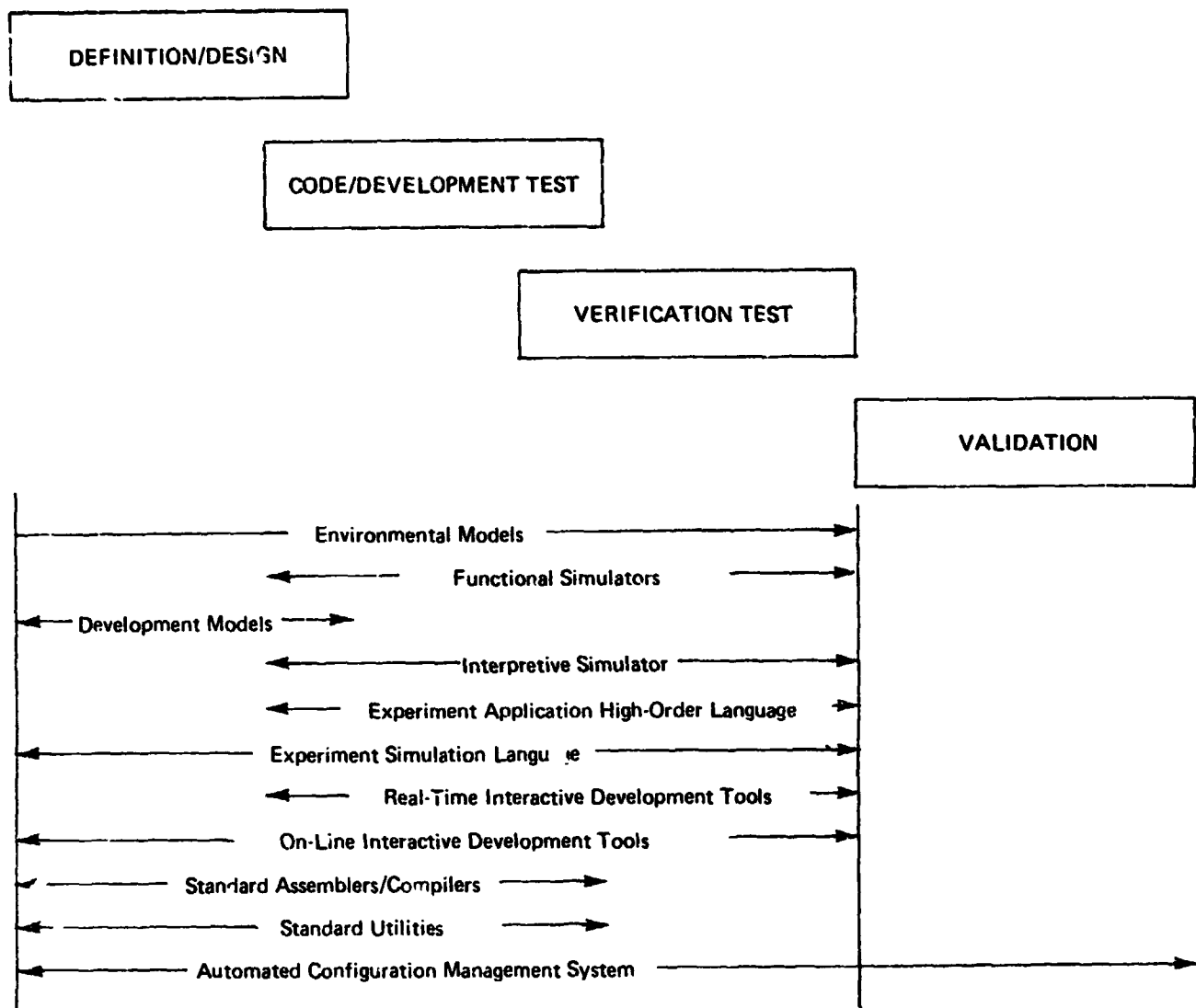


Figure 2-19. Typical Software Development Cycle with Supporting Development Tools

### Development Models

Development models are written in a HOL and are mathematical representations of the application. The models are used for concept and requirement testing during the software definition and design phase.

### CDMS Interpretive Simulators

The CDMS Interpretive Simulator, which contains a 6-D simulation of the Spacelab vehicle, is a digital model of the CDMS computing system which functions at the computer instruction level. Interpretive simulation provides a detailed logic test capability through a bit-by-bit simulation of the actual CDMS computer code. Control and perturbation of discrete, interrupt, and sensor signals are used to effect the desired logic checks. These detailed logic checks are essential in performing software verification.

### Functional Simulator

The functional simulator simulates the execution of the Experiment Flight Applications software in the language of the STIL host computer. When executed, the functional simulator performs the same functions as the software being simulated and allows testing of software concepts in near realtime.

### Experiment Application High Order Language (HOL)

The basic requirements of high productivity, fast integration, and large change activity dictates that a HOL be available for experiment application software development. The HOL provided must execute on the host computer and be capable of code generation for the experiment computer and host computer. The compiler must have built-in error detection and be cooperative with the functional simulator mode.

### Experiment Simulation Language

Because of the diversity of the experiments to be supported, a language must be provided which will allow rapid development of environment models.

### Realtime Interactive Tools

Realtime interactive software development tools provide the environment to meet the design and software development productivity requirements of Spacelab experiment development timelines. The realtime tools provide a dedicated "hands-on" environment to the software developer. Interactive tools should include the ability to dump, trace, stop and single step on the HOL statement as well as on the machine language instructions.

### Automated Configuration Management System

An Automated Configuration Management System will be required which will provide the development data base and supportive software to support multiple software configurations.

Many experiments will be reflight; however, some of the applications will be upgraded from flight to flight while others will remain stable. Availability of a module library is required to simplify the problem of assembling and integrating the software modules into the application package and Flight Set required for a particular mission. The obvious advantage of a program

library lies in the fact that software modules requiring no change can be integrated in parallel with the assembly and test of those modules requiring change.

The capability to automatically generate a release of the software system for a given payload is required. This release procedure must produce source listings, object code, tapes and required documentation as well as identify all changes to software modules and create a history of the software associated with each mission. The automatic release system must generate reports identifying the baseline as well as changes made and outstanding tasks, which may be incorporated on a priority basis, waived or held for future flights. The Automatic System Build and Release Procedure is functionally shown in Figure 2-20.

#### 2.8.3.2 Development Facilities

In previous space programs the use of a dedicated computer has proven to be an invaluable asset in the development of flight software. Within the Spacelab environment of multiple software packages in parallel development paths and with a large number of programmers to be supported, a dedicated facility is a necessity.

In development of flight software for both the Saturn and Skylab programs, an MSFC-provided facility was used. This facility was first utilized in development of the AS-507 flight program for the Saturn Launch Vehicle. As may be seen in Figure 2-21, the use of a dedicated facility improved the quality of program to such a level that the number of simulated flight hours needed for verification was reduced by more than 50%. In addition, the development cycle was reduced by approximately 50%.

To achieve the volume of output required of the Spacelab onboard experiment software development concept (unique software for each mission) a dedicated facility will be required. The facility should consist of a large scale host computer to provide the development tools needed by the programmers and should provide the capability to support realtime simulation utilizing the actual CDMS. IBM's experience on both Saturn and Skylab clearly indicates the need for testing of the operational flight software in the actual onboard computers.

Total dependence on the use of interpretive computer simulation and functional simulation tools is undesirable. As may be seen in Figure 2-22, the interpretive simulation of a Spacelab CDMS computer of 500 KOPS capability would have a run-time to flight-time of approximately 80 to 1. Within the projected development cycle for Experiment Flight Applications software, use of interpretive computer simulation for large numbers of test cases would severely impact the resources of the STIL. The functional simulation capability, although it affords realtime or faster execution, does not execute in the CDMS computer language, and thus introduces uncertainty regarding the compatibility of the Flight Applications Software and the CDMS computer.

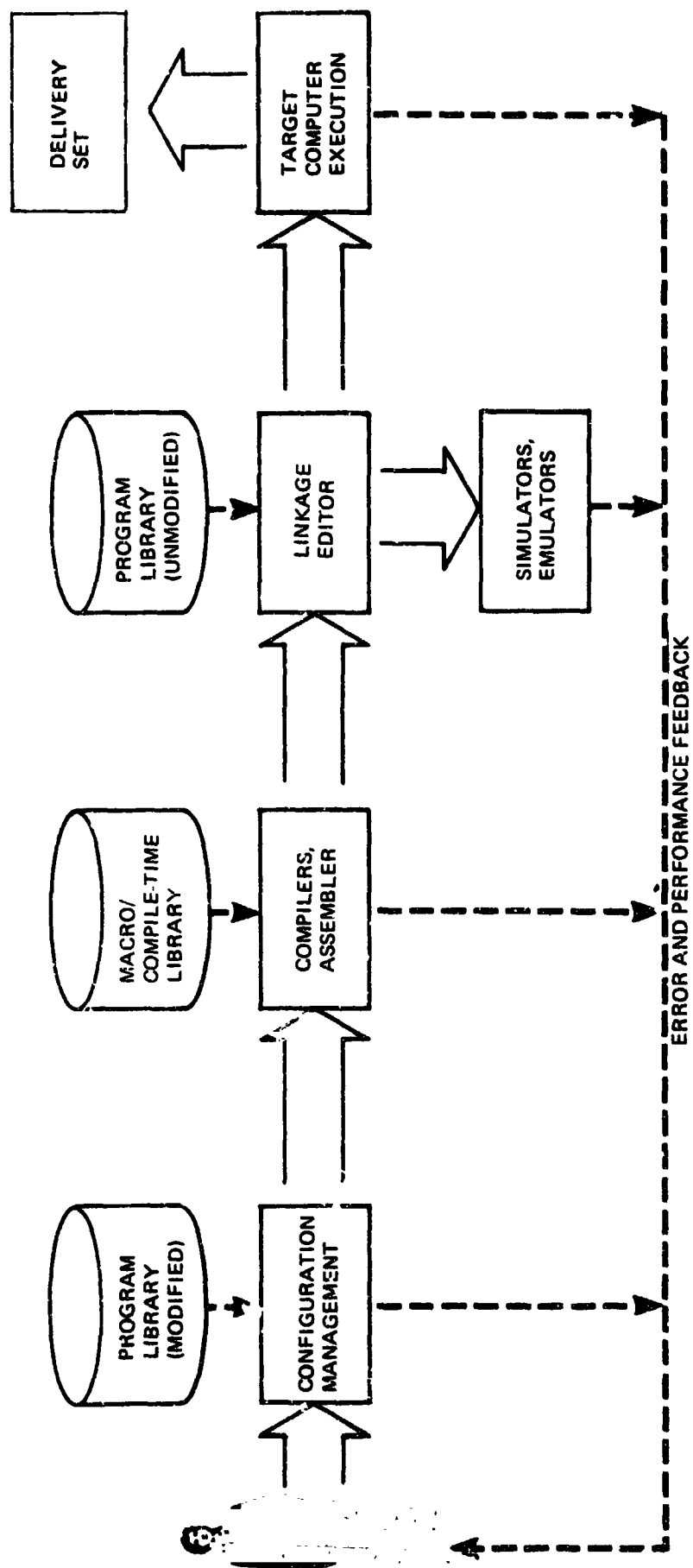


Figure 2-20. Automatic System Build and Release Flow

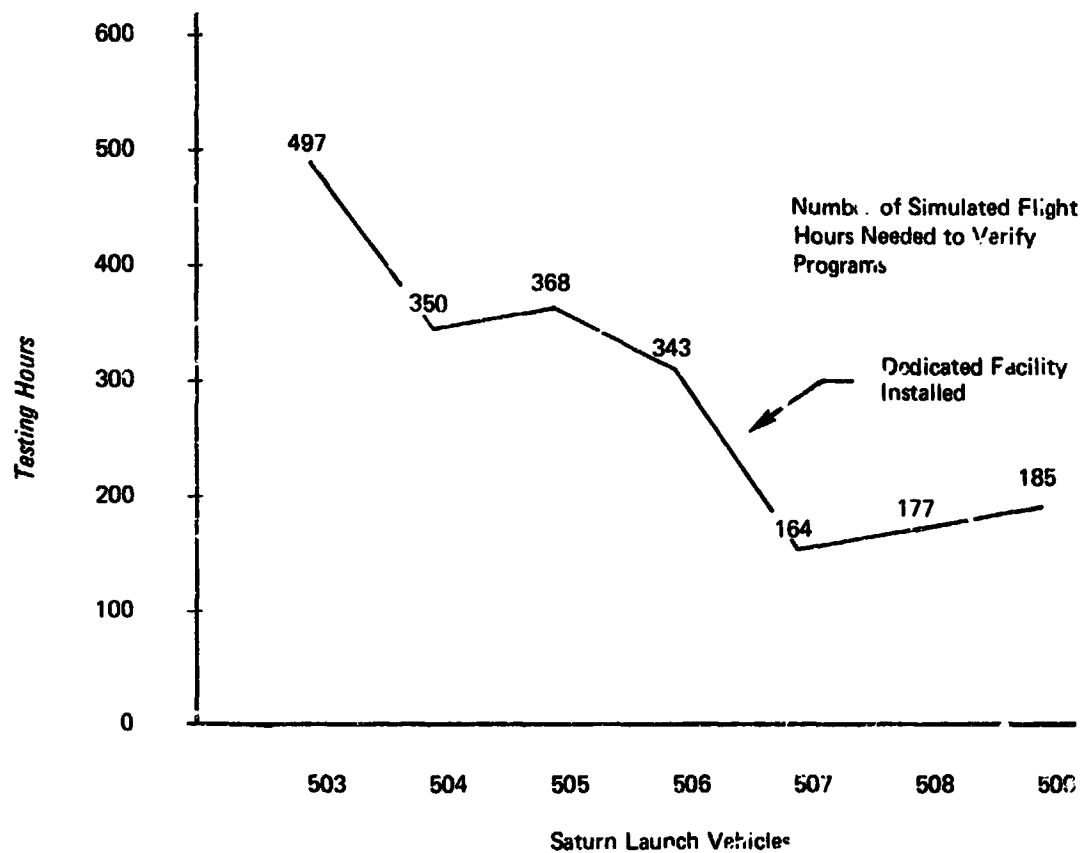
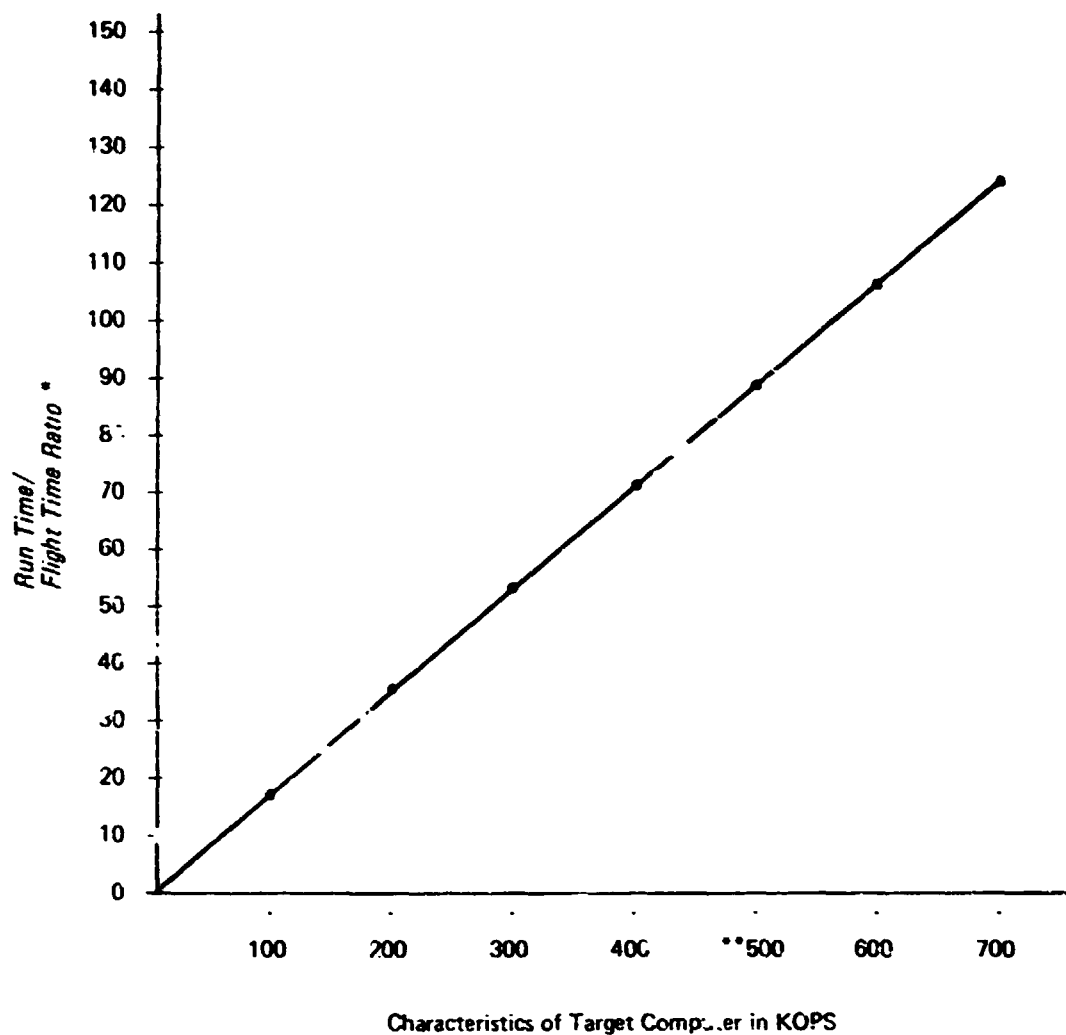


Figure 2-21. Impact of Software Development Facility on Testing



\*Based on Host Computers for Saturn IU and Skylab (IBM 360/75)

\*\*Spacelab CDMS Computer

Figure 2-27. Interpretive Computer Simulator Run Time Ratio/Flight Time Ratio



To avoid the development cycle impact of the interpretive simulation and the impact of compatibility uncertainty of the functional simulator, the capability to execute the Experiment Flight Applications software on the actual CDMS must be provided. Due to the minimum testing time environment, the simulation utilizing the CDMS must execute in realtime.

The CDMS simulation mode will require the development of a STIL host computer/CDMS Interface Device (CID). This device will provide the interface logic to support an operational environment simulation in realtime.

## 2.9 STIL REQUIREMENTS

### 2.9.1 THEME

Because of the significant software development activity associated with Experiment Flight Applications software during 1981, it is anticipated that this activity will place the most demanding processing load on the STIL. This section will address the STIL load resulting from Experiment Flight Applications software development and will provide load factors to be used for STIL modeling analysis.

### 2.9.2 CONCLUSIONS

As a result of the analysis performed in this section, the total daily load placed on STIL by flight application software development was determined to be 250 runs/day.

### 2.9.3 DISCUSSION

To establish a technical base for determining Spacelab flight application software development requirements on a development facility, studies were made of the space programs in which IBM had significant responsibility. For this base, the most recent space program, Skylab, was chosen because its flight application software was developed and verified utilizing facilities and tools very comparable to those projected for the STIL. In addition, the functions performed in Skylab flight software development (design, implementation, verification, delivery, etc.) were directly comparable to that anticipated for flight applications software. The similarities of the two programs are summarized in Table 2.8.

The results of the traffic model in Paragraph 2.2 indicate that during 1981 the STIL must support peak flight applications software development for 8 new flights and 10 reflights. This support requirement imposes a delivery of a software set every 14 calendar days to support launch schedules. For sizing studies, a development cycle of six months was chosen which corresponds closely to Skylab which also averaged six months for development. Therefore, within a development cycle period, 4 new flight packages and 5 reflight packages (a total of 9) will be simultaneously undergoing development activities during 1981.

The approach taken to develop STIL processing loads varied according to the software function. For example, the software management load was determined on a daily basis and then extrapolated into requirements for the number of packages in process. Software implementation load was computed on a module basis for compilations/assemblers/link edits and on a flight package basis for simulation test runs. The basis for software verification was test cases executed per module within the package. For software integration, the load was based on the number of flight application packages in an experiment flight set.

Table 2.8. Skylab/Spacelab Program Similarities

COMPUTER FACILITY SOFTWARE CAPABILITIES	SKYLAB		SPACELAB	SIMILARITY
	360/75	FPCF	STIL	
• MULTI-PROGRAMMING	X	X	X	X
• MULTI-TASKING	X		X	X
UTILITIES	X	X	X	X
USER AIDS		X	X	X
GRAPHICS		X	X	X
DATA REDUCTION	X	X	X	X
TERMINAL MGMT SYSTEM	X		X	X
• SIMULATORS				
REAL TIME		X	X	X
FUNCTIONAL			X	X
INTERPRETIVE	X		X	X
DESIGN ANALYSIS	X		X	X
• FLIGHT COMPUTER ASSEMBLE/LINK	X		X	X
HIGH ORDER LANGUAGE			X	
• SOFTWARE MANAGEMENT	X	X	X	X
ONBOARD SOFTWARE CAPABILITIES		SKYLAB	SPACELAB	SIMILARITY
• POINTING/CONTROL		X	X	X
• SENSOR MONITOR		X	X	X
• COMMAND & DISPLAY		X	X	X
• EXPERIMENT MONITORS			X	
• TELEMETRY		X	X	X
• OPERATING SYSTEM		X	X	X
• REDUNDANCY MANAGEMENT		X	X	X

For software implementation and software verification, a factor was needed to determine reflight package requirements based on new flight package requirements. Analysis of the instructions to be developed in 1981 showed that approximately two-fifths of the instructions would be for reflight packages. Therefore, the two-fifths factor was applied to new flight data to estimate reflight package development requirements on STIL.

As was noted previously, the number of anticipated Spacelab program modules was required to develop load factors. To estimate the number of modules that will make up a typical flight applications software package, a factor must be applied to the number of assembler instructions. By analyzing the current compiler outputs at the IBM computer center, a ratio of 5:1 assembler instructions to high order language statements was determined. Since 35,600 assembler instructions typify a flight package, the 5:1 factor yields 7,200 high order language statements. Applying the precepts of structured programming to flight application software development restricts the source statements to an average of 100 per module. This indicates that the typical Spacelab flight application will require 72 modules.

The summarization tables in the following discussions will have columnar headings of Package/Set, Cycle and Day. These descriptions identify the anticipated requirements as follows:

- Package/Set - indicates the number of runs for a single new flight or reflight.
- Cycle - indicates the number of runs for all new flights or reflights during the 6-month period.
- Day - indicates the number of runs for all new flights or reflights for a single day.

#### 2.9.3.1 STIL Requirements for Experiment Flight Applications Software

The functions which must be performed in the STIL to generate quality flight applications software within the development concept were determined to include:

- Software management
- Software implementation
- Software verification
- Software integration

### Software Management

The STIL host computer will be utilized as a management tool to track all development activity and to provide an automated method of generating the necessary software delivery data. This type processing is mandatory when considering the delivery of nine CDMS flight sets in a 6-month period.

To maintain configuration control over flight applications software development, it has been assumed that one run per day is required to maintain the software development activity data base. On a weekly basis, reports must be generated for tracking of problem reports and change activity for all flight software modules and packages.

Past experience indicates that a minimum of three package releases will be made for each application--two preliminary releases (for verification/error correction, training, etc.), and the final release.

The summary of STIL load resulting from software management is shown in Table 2.9.

### Software Implementation

The following STIL capabilities will be utilized while developing flight applications software:

- CDMS computer assembler/compiler/linkage editors
- Interpretive simulator
- Functional simulators
- Realtime simulators
- Design analysis simulators
- Data reduction

For establishing the projected STIL utilization, the Phase II Skylab implementation data was used because of its comparable development cycle and the availability of accurate utilization figures.

Compiles/Assembles/Link Edits - To estimate compiles/assembles/link edits per package, the Skylab base of 4.67 assembles/link edits per day over a 180-day period was used. This base yields 840 (4.67 x 180) total runs. Since Space Lab analysis has shown that approximately 72 modules will be required versus 49 for the Skylab program, an estimate of a 50% increase in the number of assembles/compiles/link edits, or 1,260 runs, was established for Spacelab.

**Table 9. Summary of Software Management Load on STIL**

SOFTWARE MANAGEMENT	RUNS/DAY					
	NEW FLIGHTS			REFLIGHTS		
	PACKAGE	CYCLE	DAY	PACKAGE	CYCLE	DAY
Configuration Management	130	520	4	130	650	5
Statistics	26	104	0.8	26	130	1.0
Automated Release	3	12	0.09	3	15	.12
SUBTOTALS	159	636	4.9	159	795	6.1
OVERALL TOTAL/DAY = 11 RUNS						

Interpretive Simulator - An interpretive simulator was not used extensively in the Skylab development because of run ratios of processing time to flight time (80:1). To supply a load on STIL for ICS utilization, an assumption of one run per day over the 6-month development cycle was made. This results in 130 runs for new flight operation packages.

Functional Simulator - Since Skylab was written in assembler language, a functional simulator was not provided. However, the realtime simulation mode was used approximately two-thirds of the time in Skylab development for the low fidelity simulation characteristics of a functional simulation. Reduction of Skylab realtime runs (810) by the two-thirds factor yields an estimated 540 runs for equivalent functional simulation on new flight packages.

Realtime Simulation - During Phase II Skylab program implementation, there were 270 realtime simulation runs required for package testing. Similarly, Spacelab flight applications software will require approximately 270 realtime CDMS simulation runs for new flight packages.

Design Analysis Simulation - There were 180 design analysis simulation runs made with this simulation tool during Phase II Skylab implementation; thus, Spacelab will also require 180 runs per package for new flight packages.

Data Reduction - For data reduction load factor determination, it was assumed that 75% of the above simulations required data reduction runs to provide complete data. For software implementation the number of simulation runs was 1,120. Data reduction, computed as 75% of the composite simulation runs, yields a total of 840 runs for each new flight package.

The total load on the STIL for program implementation is summarized in Table 2.10. The run data for reflight loading is based on two-fifths of a new flight application and, as can be seen, a total of 151 runs per day for implementation will be required.

#### Software Verification

Verification activity is the bridge between specification and implementation. This activity polices the programmer's coded implementation and the PI's performance specification. Based on previous NASA flight software development experience, in conjunction with the anticipated burden of nine programs in various stages of development during 1981, the verification/integration of Spacelab programs is anticipated to be the major user of STIL resources.

**Table 2.10. Summary of Software Implementation Requirements on STIL**

SOFTWARE IMPLEMENTATIONS	RUNS/DAY					
	NEW FLIGHT			REFLIGHT		
	PACKAGE	CYCLE	DAY	PACKAGE	CYCLE	DAY
CDMS Computer Assem- ble/Compile/Link Edit	1260	5040	39	504	2520	19
Interpretive Simulation	130	520	4	52	260	2
Functional Simulation	540	2160	7	216	1080	9
CDMS Simulation	270	1080	9	108	540	4
Design Analysis Simu- lation	180	720	6	72	360	3
Data Reduction	840	3360	26	336	1680	13
SUBTOTALS	3220	12,880	101	1288	6440	50
TOTAL RUNS/DAY = 151						



The approach used to estimate verification requirements on STIL was to determine the number of test runs per program module for Skylab and then, based on the number of modules in a Spacelab new flight application package, extrapolate to generate the test case load.

Verification of flight applications software on the STIL will require the following capabilities:

- Interpretive computer simulation
- Realtime simulation
- Data reduction

Interpretive Computer Simulator (ICS)- Skylab utilized the ICS for 441 test cases. On a test/module basis, this resulted in nine tests per module. Spacelab, with 72 modules/package, will, therefore, require 648 test cases for verification on an ICS for new flights. Applying the retest factor of two-fifths for reflight packages yields 259 test cases per reflight package.

Realtime Simulation - The CDMS realtime simulation provides unique verification aids for testing in a high fidelity simulated environment using actual flight computers. Realtime simulation test cases for Skylab totaled 291 or 7 per module. Spacelab requirements, based on 72 modules, yields 432 test cases for new flight packages and 173 for reflights.

Data Reduction - As was done previously for implementation, it was assumed that 75% of all simulations required data reduction processing. For verification, there is a total of 1,080 simulations required for a Spacelab new mission. Applying the 75% data reduction factor results in 810 data reduction runs for new flight packages.

Table 2.11 represents a summary of flight applications software verification requirements on STIL. As can be seen, a total of 88 runs per day will be required for verification.

#### Software Integration

Integration of the flight application packages into CDMS flight sets represents the final testing prior to delivery. For Spacelab, this activity is similar to the Skylab activity performed at the MSFC Hybrid Simulation Laboratory (HSL). The HSL provided the capability to exercise the onboard computer software in a realtime hardware environment. Thus, the HSL provided a test bed for the onboard software much like the actual ATM hardware. In order to avert malfunction downtime on the actual hardware, software models of the hardware were developed as a backup.

**Table 2.11. Summary of Software Verification on STIL**

SOFTWARE VERIFICATION	RUNS/DAY					
	NEW FLIGHT			REFLIGHT		
	PACKAGE	CYCLE	DAY	PACKAGE	CYCLE	DAY
Interpretive Computer Simulation	648	2592	20	259	1295	10
Realtime Simulation	432	1728	13	173	865	7
Data Reduction	810	3240	25	324	1620	13
SUBTOTALS	1890	7560	58	756	3780	30
TOTAL RUNS/DAY = 88						

Limited time is provided in the CIS for integration testing. This places a burden on the final STIL integration testing to ensure correct interfaces and thus allow minimum hardware/software integration testing time at the CIS.

Anticipatory to problems during system build and final test, it has been assumed that two computer runs for each system build and final test for each mission will be required. Table 2.12 represents the summary of software integration load requirements on the STIL for new flight sets and reflight sets.

#### 2.9.3.2 Summary of STIL Requirements

The total impact of flight applications software development on the STIL is summarized in Table 2.13. Note that certain tasks, such as automated release, are performed infrequently and result in only a fractional impact per day. However, since the STIL model discussed in Section 5.4 is based on daily load, the approach taken in this section was to carry fractional loads if necessary. As can be seen, a daily load of 250 runs will be required to support flight applications software development.

*Table 2.12. Summary of Software Integration Load on STIL*

SOFTWARE INTEGRATION	RUNS/DAY					
	NEW FLIGHTS			REFLIGHT		
	SET	CYCLE	DAY	SET	CYCLE	
Software Set Build	2	8	.06	2	10	.07
Integrated Testing Set	2	8	.06	2	10	.08
SUBTOTALS	4	16	.12	4	20	.16
TOTAL RUNS/DAY = .28						

Table 2.13. Summary STIL Requirements from Flight Application Development

FLIGHT APPLICATION FUNCTION	RUNS/DAY					
	NEW FLIGHTS			REFLIGHTS		
	PACKAGE/SET	CYCLE	DAY	PACKAGE/SET	CYCLE	DAY
<b>SOFTWARE MANAGEMENT</b>						
● Configuration Mgmt	130	520	4.0	130	650	5
● Statistics	26	104	0.8	26	130	1
● Automated Release	3	12	.09	3	15	.12
<b>SOFTWARE IMPLEMENTATION</b>						
● CDMS Computer, Compile/Assemble/ Link	1260	5040	39	504	2520	19
● Interpretive Sim.	130	520	4	62	260	2
● Functional Sim.	540	2160	17	216	1080	9
● CDMS Simulator	270	1080	9	108	540	4
● Design Analysis Sim.	180	720	6	72	360	3
● Data Reduction	840	3360	26	336	1680	13
<b>SOFTWARE VERIFICATION</b>						
● Interpretive Sim.	648	2592	20	259	1035	10
● CDMS Simulator	432	1728	13	173	865	7
● Data Reduction	810	3240	25	324	1620	13
<b>SOFTWARE INTEGRATION</b>						
● System Set Build	2	8	.06	2	10	.08
● Software Set Inte- gration Test	2	8	.06	2	10	.06
<b>SUBTOTALS</b>	<b>5273</b>	<b>21,092</b>	<b>164.01</b>	<b>2207</b>	<b>11,035</b>	<b>86.3</b>
<b>TOTAL RUNS/DAY = 250</b>						

# TASK 3B: IDENTIFY THE SPACELAB TEST AND CHECKOUT SOFTWARE CONCEPTS

## 3

SECTION		PAGE
3	TASK 3B: IDENTIFY THE SPACELAB TEST AND CHECKOUT SOFTWARE CONCEPTS . . . . .	3-1
3.1	TASK 3B: SUMMARY . . . . .	3-1
3.1.1	Objectives/Study Approach . . . . .	3-1
3.1.2	Test and Checkout Software Definition . . . . .	3-3
3.1.3	Levels of Test and Checkout Software Participation in Testing . . . . .	3-3
3.2	ANALYSIS OF SPACELAB TEST AND CHECKOUT FLOW . . . . .	3-5
3.2.1	Theme . . . . .	3-5
3.2.2	Conclusions . . . . .	3-5
3.2.3	Discussion . . . . .	3-5
3.2.3.1	Engineering Model Phase . . . . .	3-6
3.2.3.2	Initial Phase . . . . .	3-8
3.2.3.3	Operational Phase . . . . .	3-8
3.2.3.4	Test and Checkout Flow Summary . . . . .	3-10
3.3	TEST AND CHECKOUT SOFTWARE CONCEPTS . . . . .	3-13
3.3.1	Theme . . . . .	3-13
3.3.2	Conclusions . . . . .	3-13
3.3.3	Discussion . . . . .	3-13
3.3.3.1	Ground Checkout Software Capability . . . . .	3-13
3.3.3.2	On-Orbit Test and Checkout Capability . . . . .	3-17
3.3.3.3	Test and Checkout Software Utilization . . . . .	3-20
3.3.3.4	On-Orbit Test and Checkout Utilization . . . . .	3-25
3.4	TEST AND CHECKOUT SOFTWARE DESIGN CONSIDERATIONS . . . . .	3-27
3.4.1	Theme . . . . .	3-27
3.4.2	Conclusions . . . . .	3-27
3.4.3	Discussion . . . . .	3-27
3.5	TEST AND CHECKOUT SOFTWARE MAINTENANCE REQUIREMENTS . . . . .	3-31
3.5.1	Theme . . . . .	3-31
3.5.2	Conclusions . . . . .	3-31
3.5.3	Discussion . . . . .	3-31
3.6	STIL REQUIREMENTS . . . . .	3-37
3.6.1	Theme . . . . .	3-37
3.6.2	Conclusions . . . . .	3-37
3.6.3	Discussion . . . . .	3-37
3.6.3.1	STIL Requirements for Test and Checkout Software . . . . .	3-37
3.6.3.2	Summary of Test and Checkout Impact on STIL . . . . .	3-40

# IDENTIFY THE SPACELAB TEST AND CHECKOUT SOFTWARE CONCEPTS 3

## 3.1 TASK 3B: SUMMARY

In support of the Spacelab Hardware Development and Operational phases, Test and Checkout Software will play a significant role in all levels of testing. This produces a significant impact on the overall Spacelab program, therefore, a systematic approach to providing the software to support test and checkout capabilities must be developed. This approach must emphasize commonality of software and hardware across many applications of the Spacelab to reduce development costs while at the same time providing flexibility and growth potential to cover the Spacelab lifetime.

### 3.1.1 OBJECTIVES/STUDY APPROACH

The objectives of the Spacelab Test and Checkout software concept analysis were to:

- Identify Test and Checkout software requirements.
- Define an overall Test and Checkout software concept for the Spacelab Program.
- Define NASA/ESRO responsibilities in relation to Test and Checkout software.
- Determine the impact of Test and Checkout software on the STIL.

The approach utilized in the Test and Checkout software concepts analysis is shown in Figure 3-1. The analysis was conducted in five phases:

- Analysis of the Spacelab Test and Checkout flow to determine Test and Checkout software requirements and NASA software support requirements.
- Development of the overall Spacelab Test and Checkout software concepts.
- Determination of Test and Checkout software design considerations.
- Identification of level of maintenance activity required and the software development tools and facilities needed to provide the maintenance capability.
- Determination of the STIL support requirements.

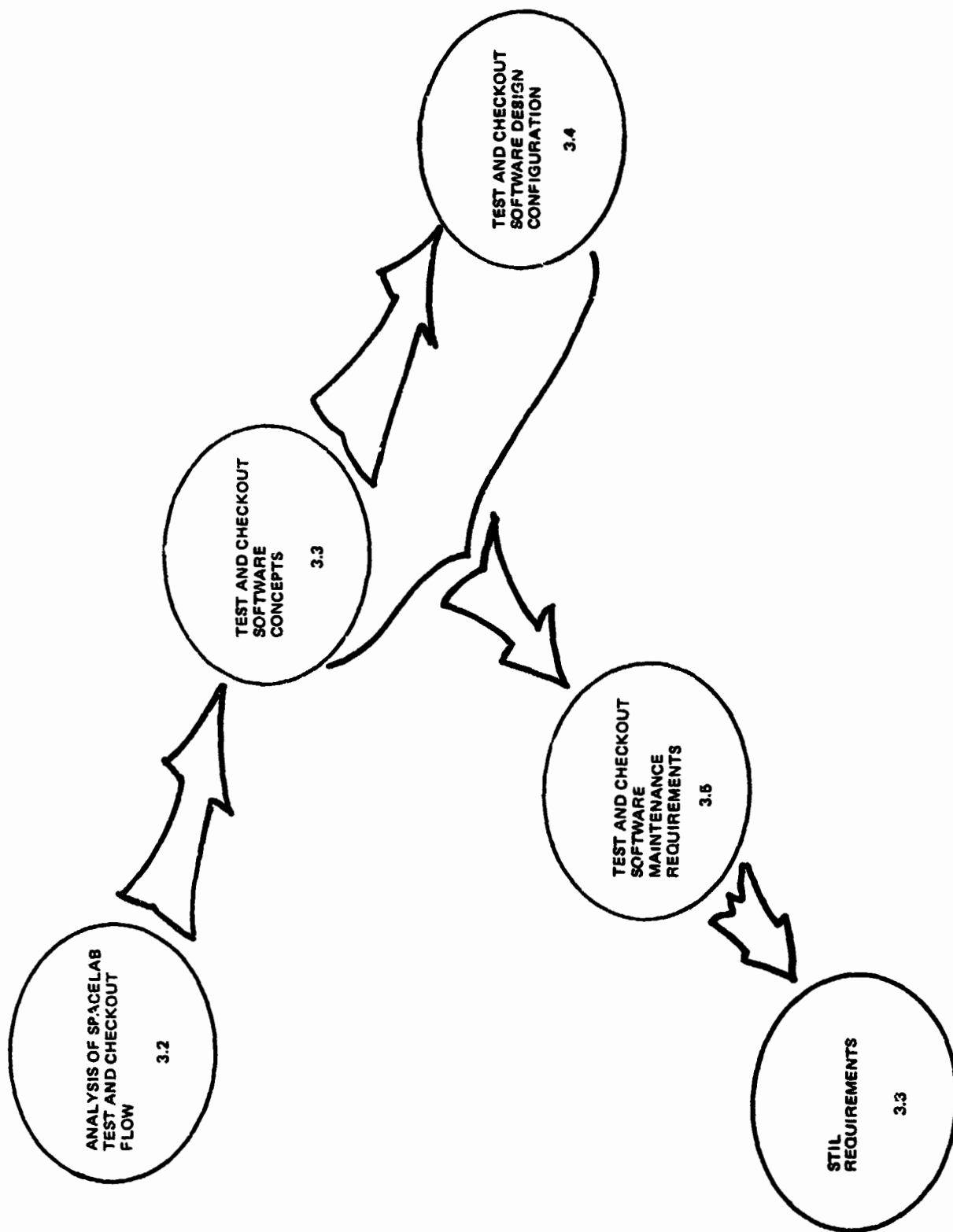


Figure 3-1. Test and Checkout Software Concepts Study Approach

The discussion of these analysis phases will be provided in paragraphs indicated by the numbers in Figure 3-1.

### 3.1.2 TEST AND CHECKOUT SOFTWARE DEFINITION

Within this study, Test and Checkout software has been defined as that software within the CDMS and EGSE computers required to support ground checkout and within the CDMS computers to support on-orbit test and checkout.

#### Ground Checkout Software

The ground checkout software set will consist of the EGSE ground checkout set plus a CDMS ground checkout set. The EGSE ground checkout set will include EGSE software needed to support all Spacelab ground checkout between initial integration through refurbishment.

The CDMS ground checkout set will consist of Test and Checkout software packages for both the experiment and subsystem computers. To ensure overall Spacelab onboard system integrity, these packages will provide the special testing software needed during Spacelab integration and pre-launch testing.

#### On-Orbit Test and Checkout Software

The On-Orbit Test and Checkout software will be comprised of all the software associated with the subsystem computer and that portion of the experiment computer which is dedicated to the test and checkout of the experiments and the Experiment Data Management System.

### 3.1.3 LEVELS OF TEST AND CHECKOUT SOFTWARE PARTICIPATION IN TESTING

Test and Checkout software is a major element of the hardware integration philosophy for Spacelab. It is anticipated that the Spacelab hardware will be test compatible with the software to provide maximum onboard test and checkout capability and allow the software to direct failures and isolate failed components to the Lowest Replaceable Unit (LRU). To provide the test and checkout capability, software must be utilized at all integration testing levels.

The present definition of Spacelab hardware integration requires four distinct levels:

Level IV - Instrument Assembly Integration

Level III - Experiment/Experiment Module Integration

Level II - Support Module/Experiment Module/Pallet Integration

Level I - Spacelab/Shuttle Integration



In support of integration Levels I, II, and III, the Test and Checkout software must provide the varying capabilities necessary to perform the required testing. Level IV integration is considered a responsibility of the experiment PI during manufacture and is not considered a portion of the Test and Checkout software concept.

In addition to supporting the above integration levels, the Test and Checkout software must perform on-orbit testing of the Spacelab subsystem and experiments. Implicit within the support requirement is the participation of the Shuttle Orbiter and Payload Operations Center.

## 3.2 ANALYSIS OF SPACELAB TEST AND CHECKOUT FLOW

### 3.2.1 THEME

In order to establish the Test and Checkout software concepts for Spacelab, an understanding of the hardware development and integration flow is required. This understanding results in the identification of the varying levels of testing, facilities, and test equipment to be utilized. Based on the levels of testing, a set of Test and Checkout software requirements can be established.

### 3.2.2 CONCLUSIONS

Based on the analysis of the overall hardware testing requirements, the following conclusions relative to Test and Checkout software and facilities have been determined:

- Hardware integration testing will require CDMS and EGSE Test and Checkout software.
- For NASA to maintain the Test and Checkout software, the STIL must provide development tools and testing capabilities.
- The STIL must be operational prior to delivery of the Engineering Model in first quarter of 1978.
- ESRO must make available CDMS/EGSE support software and models to STIL prior to shipment of the Engineering Model.
- ESRO will provide CDMS and EGSE computer software for the Engineering Model and first two flight units.

### 3.2.3 DISCUSSION

The analysis of the Spacelab ground operations plan established three distinct hardware development, integration, and testing phases. These have been defined in the following manner:

Engineering Model Phase contains the activities associated with NASA participation in acceptance, testing, and installation of the Engineering Model within NASA testing facilities.

Initial Phase contains those activities associated with NASA participation in the flow of the first two flight units.

Operational Phase contains those activities required to support hardware development, integration, and testing after the first two flight units.

Each of the above phases will result in requirements being established for the Test and Checkout software concept. For this reason, each of the phases will be discussed in the following paragraphs. For the flow within each phase, the facilities utilized will be identified and the Test and Checkout software requirements established.

#### 3.2.3.1 Engineering Model Phase

Prior to delivery of the Spacelab flight unit hardware, ESRO will deliver an Engineering Model (EM) of the Spacelab hardware to MSFC. The purpose of the EM will be to facilitate integration site activation at MSFC and KSC. Following site activation exercises, the model will be returned to MSFC for use as an engineering test bed during the initial and operational phases.

The flow of the EM with associated facilities, deliverable and utilization is shown in Figure 3-2. As may be seen, the EM is developed by ESRO in Europe and provides the following hardware and software deliverables related to Test and Checkout software:

- EGSE Computer
- Automatic Test Equipment (ATE)
- Simulators
- CDMS
- CDMS Flight Set
- EGSE Software
- CDMS Subsystem Ground Checkout Set
- CDMS Experiment Ground Checkout Set

Upon receipt at MSFC, the ESRO EM software will be utilized for activation of the MSFC Software Test and Integration Laboratory (STIL). It has been assumed during this study the ESRO will make available the support software for the CDMS and EGSE computers prior to delivery of the EM. This early delivery will be required in order to ensure that the STIL can be utilized for EM software integration and testing with a minimum of problems.

The facilities to be activated through use of the EM are:

- Software Test and Integration Laboratory (STIL)
- Central Integration Site (CIS)
- Manned Spacecraft Operations Building (MSOB)

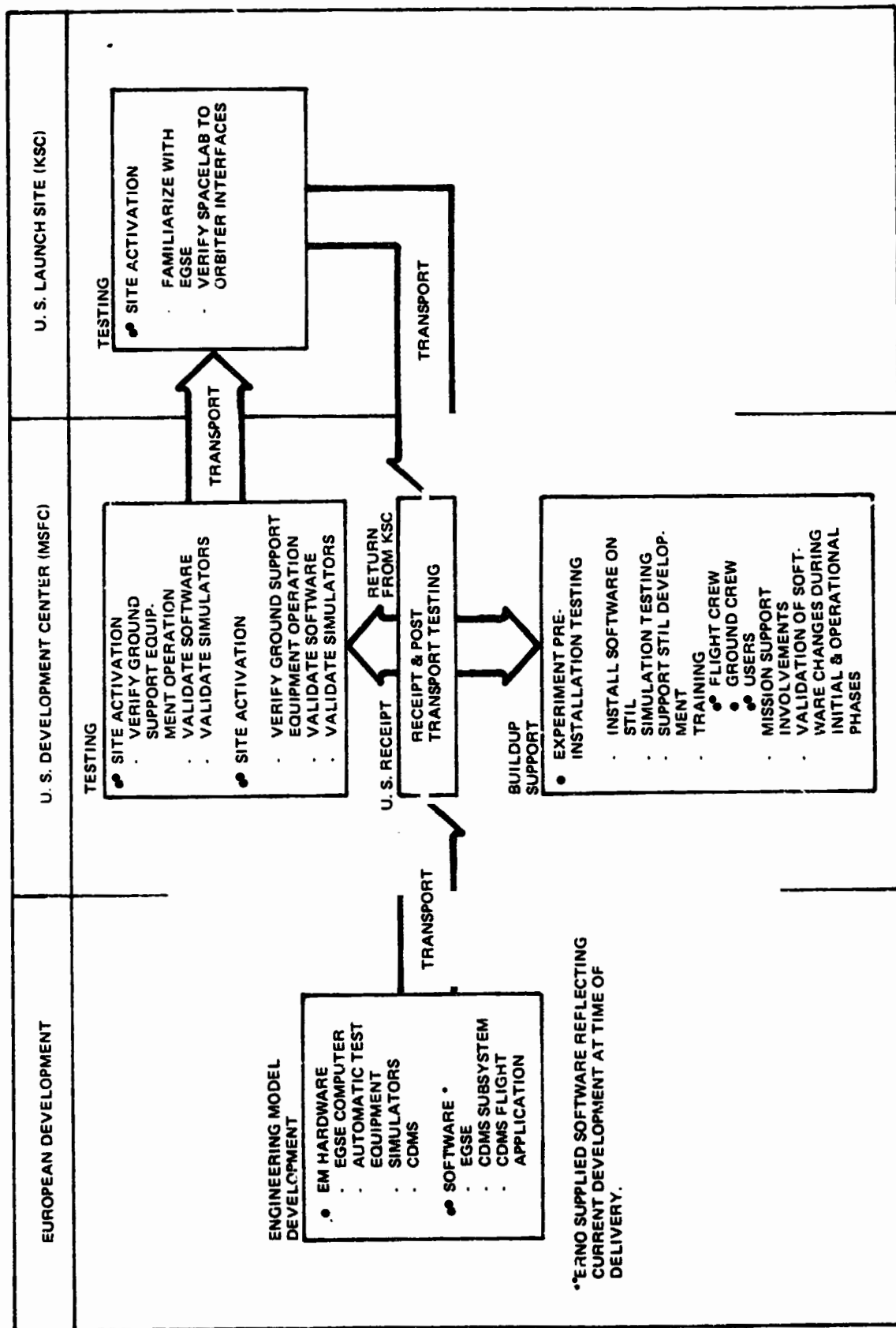


Figure 3-2. Spacelab Engineering Model Flow

At each facility, the Test and Checkout hardware and software will serve the dual purposes of performing test and checkout of the EM and the training of test personnel.

#### 3.2.3.2 Initial Phase

The initial phase of the flow includes those activities required in support of the first two flight units. As was the case with the EM, ESRO will develop both the hardware and software for the first two flight units and will perform integration testing on the total Spacelab systems prior to shipment to MSFC.

The flow of hardware and software during the initial phase is shown in Figure 3-3. To support this flow, it will be required that all NASA facilities be fully operational. It should be noted that upon return from orbit, the hardware will be refurbished, as necessary, and placed into storage for use on subsequent missions.

Although ESRO will develop all Test and Checkout software during this phase, the STIL will be required to be fully operational. The ESRO-provided software for the CDMS and EGSE will be installed on the STIL, and all support functions provided by the STIL will be exercised. Within this environment, the STIL must provide the following capabilities:

- Operational Support Software for CDMS and EGSE Computers
- Full Testing Capability for all CDMS and EGSE Software
- Software Management Tools for Configuration Control
- Capability to Generate CDMS Ground Checkout Sets, CDMS Flight Sets, and EGSE Ground Checkout Sets

As a verification medium for the overall STIL capabilities, the software sets generated on the STIL will be compared to the corresponding sets generated by ESRO for the flight units. Upon completion of the initial phase all STIL capabilities needed in support of Spacelab CDMS and EGSE software development will have been verified for use in the operational Spacelab environment.

#### 3.2.3.3 Operational Phase

The operational phase will be utilized for all flight units subsequent to the initial two units. Within this phase, NASA will maintain total responsibility for the Spacelab hardware and software utilized in performing test and checkout and will perform all integration and testing. The operational flow will differ from the initial flow in the following ways:

- Spacelab hardware and software, previously utilized, will exist in storage rather than being delivered by ESRO.

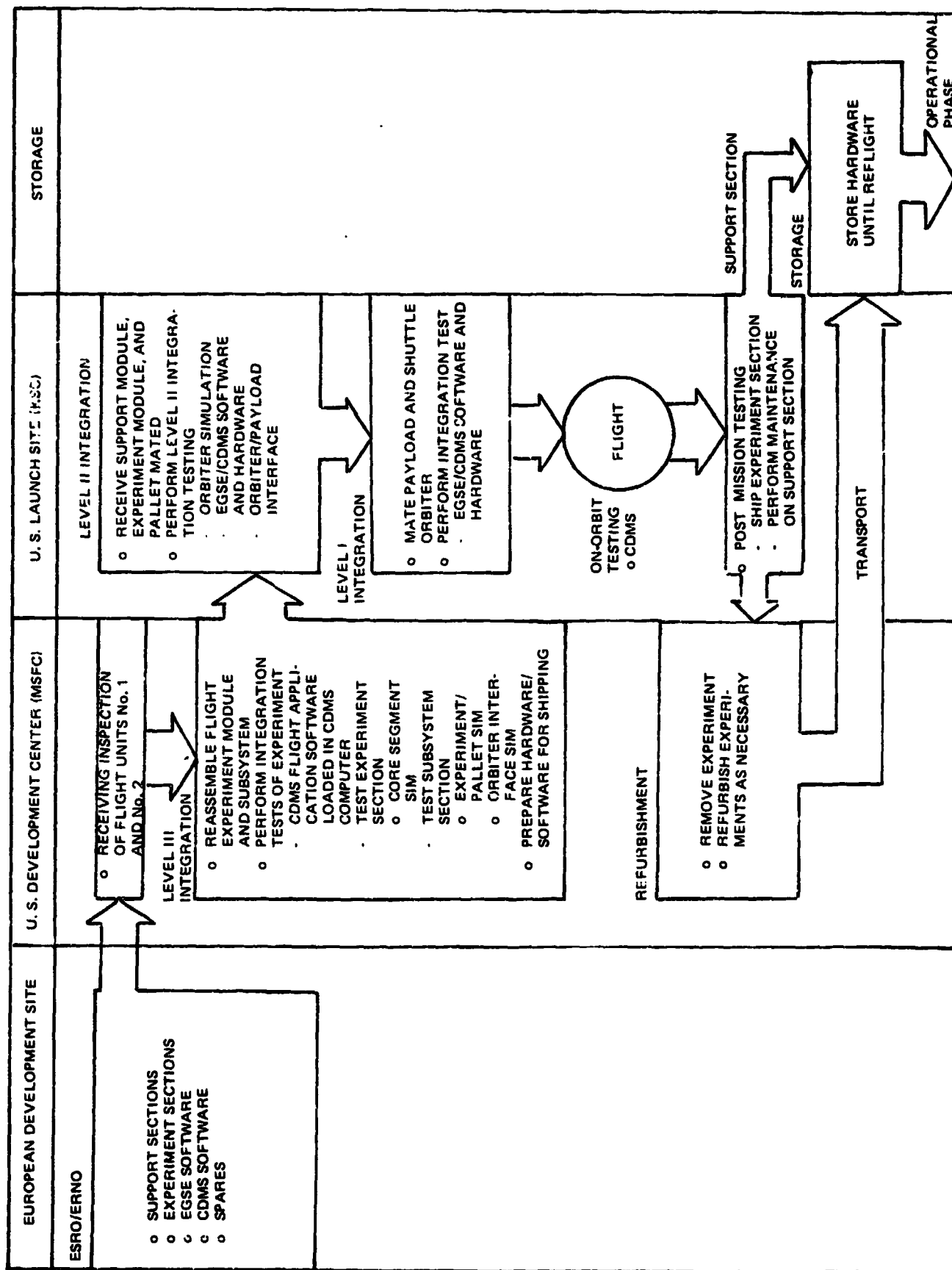


Figure 3.3. Spacelab Hardware/Software Initial Unit Flow

- Experiment Flight Applications software will be either developed on the STIL or will be delivered to MSFC for testing and integration into CDMS flight sets.
- Experiment hardware will be developed by the PI and delivered to MSFC for integration testing (Level III) at the CIS.
- NASA will maintain and update test and checkout software, as required, through the capabilities of the STIL.

The flow to be utilized during the operational phase is shown in Figure 3-4. Within this study, the assumption has been made that refurbishment of the support section (repair by replacement of identical hardware) will be accomplished at KSC; however, refurbishment (implementation of engineering changes) of the support section will require that the support section be returned to the CIS for modification and recertification.

#### 3.2.3.4 Test and Checkout Flow Summary

A summary of the results of the test and checkout flow analysis is shown in Table 3.1.

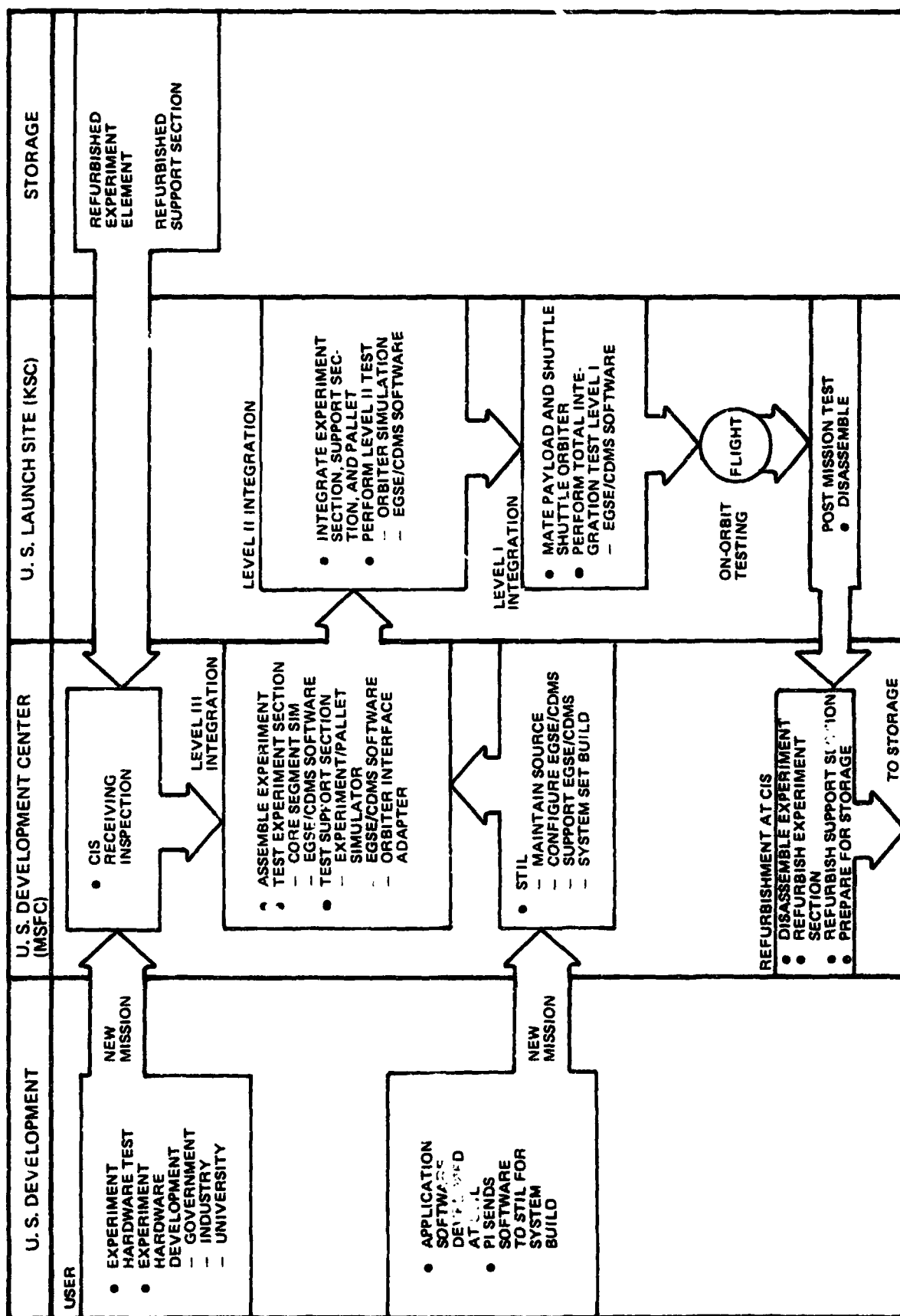


Figure 3-4. Space Shuttle Hardware/Software Operational Unit Flow



Table 3.1. Summary of Spacelab Ground Operations Analysis

SPACELAB ITEM	SOFTWARE		LEVEL III CIS	LEVEL III MSOB	LEVEL I LAUNCH SITE	RESPONSIBLE ORGANIZATION
	ESRO STIL	NASA STIL				
<ul style="list-style-type: none"> <li>Engineering Model</li> </ul>	<ul style="list-style-type: none"> <li>Development</li> <li>Fit Application</li> <li>EGSE</li> <li>Subsystem</li> <li>Simulators</li> <li>Functional</li> <li>Interpretive</li> <li>Design</li> <li>Data Reduction</li> </ul>	<ul style="list-style-type: none"> <li>Install</li> </ul>	<ul style="list-style-type: none"> <li>Site Activation</li> <li>Integrated Exp Section Test</li> <li>EGSE/CDMS Software</li> <li>Core Segment Orbiter Simulator</li> <li>Integrated Support Section Test</li> <li>EGSE/CDMS Software</li> <li>Exp./Pallet Simulator</li> <li>Orbiter Interface Adapter</li> </ul>	<ul style="list-style-type: none"> <li>Site Activation</li> <li>Integrated Spacelab Test</li> <li>EGSE/CDMS Software</li> <li>Orbiter Simulator</li> </ul>		<ul style="list-style-type: none"> <li>ESRO</li> </ul>
<ul style="list-style-type: none"> <li>Flight Unit</li> </ul>	<ul style="list-style-type: none"> <li>Revised Software</li> </ul>	<ul style="list-style-type: none"> <li>Install Revised Software</li> </ul>	<ul style="list-style-type: none"> <li>Integrated Exp Section Test</li> <li>Same as Above</li> <li>Integrated Support Section Test</li> <li>Same as Above</li> </ul>	<ul style="list-style-type: none"> <li>Integrated Spacelab Test</li> <li>Same as Above</li> </ul>	<ul style="list-style-type: none"> <li>Spacelab/Orbiter Integration Test</li> <li>CDMS Software</li> <li>Communications</li> </ul>	<ul style="list-style-type: none"> <li>ESRO/NASA</li> </ul>
<ul style="list-style-type: none"> <li>Operational</li> </ul>		<ul style="list-style-type: none"> <li>Maintain EGSE AND CDMS Subsystem Software</li> <li>Develop Experiment Flight Application Software</li> </ul>	<ul style="list-style-type: none"> <li>Refurbish</li> <li>Exp. Section</li> <li>Support Section</li> <li>Integrated Test</li> <li>Exp. Section</li> <li>Support Section</li> <li>Same as Above</li> </ul>	<ul style="list-style-type: none"> <li>Integrated Spacelab Test</li> <li>Same as Above</li> </ul>	<ul style="list-style-type: none"> <li>Spacelab/Orbiter Integration Test</li> <li>Same as Above</li> </ul>	<ul style="list-style-type: none"> <li>NASA</li> </ul>

### 3.3 TEST AND CHECKOUT SOFTWARE CONCEPTS

#### 3.3.1 THEME

The Test and Checkout software concepts to be utilized for Spacelab testing must support the planned hardware flow and must provide the required testing capabilities for all levels of integration of Spacelab hardware. To establish the Test and Checkout software concepts for Spacelab, the capabilities of the hardware in support of testing is a key element.

#### 3.3.2 CONCLUSIONS

The following conclusions have been established as a result of this analysis:

- Provisions must exist within onboard CDMS hardware to allow software testing.
- EGSE must include stimulus generators and special measuring devices to support detailed testing.
- A standard EGSE computer, with test conductor CRT control, will be required for all levels of integration testing.
- CDMS computer software must operate under control of EGSE software to provide required testing capability.
- Simulators developed by ESRO will be required in achieving testing objectives.
- Extensive man/machine interface capability will be required within the EGSE.

#### 3.3.3 DISCUSSION

The development of Test and Checkout software concepts requires an understanding of the software sets to be utilized as well as the method of utilization to meet the Spacelab testing requirements. To achieve this understanding, the following paragraphs will discuss, first, the capabilities of the CDMS and EGSE ground checkout software sets and their application to establishing the overall test and checkout concepts.

##### 3.3.3.1 Ground Checkout Software Capability

As indicated in Figure 3-5, the ground checkout capability will be provided through the CDMS and EGSE ground checkout sets required to support the test and checkout requirements. The testing capabilities of these sets will be discussed in the following paragraphs.

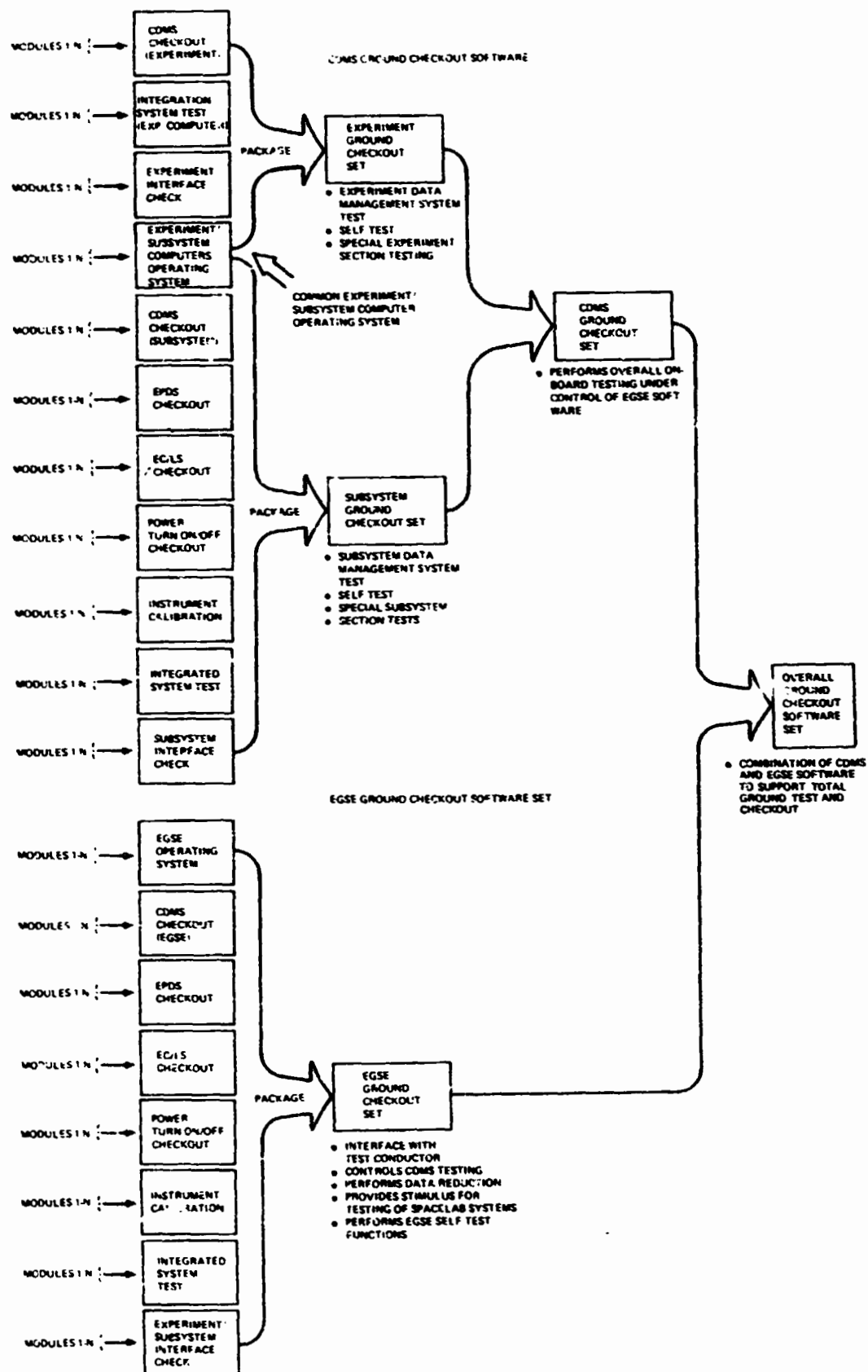


Figure 3-5. Ground Checkout Software

### CDMS Ground Checkout Set

The key elements in the performance of test and checkout of the Spacelab onboard systems are the software, which resides in the CDMS computers, and Built-In Test Equipment (BITE) within the hardware. The BITE will provide the capability of maximizing the use of CDMS software for hardware testing and will assist in achieving the goal of minimizing ground test equipment during testing.

- Subsystem Ground Checkout Set - The subsystem ground checkout set will provide the capability to perform test and checkout of the subsystem data management system elements as well as perform test and checkout of the subsystems.

For test and checkout of the subsystem data management system, the test and checkout software will test the following elements as indicated.

Subsystem Computer testing will be achieved through the self-test software package resident within the computer.

Subsystem I/O testing will be achieved through I/O parity capability between the computer and the subsystem I/O device.

Mass Memory testing will require the software to read a fixed memory load into the computer and perform checksums to ensure correct operation.

CRT/Keyboard testing will require interaction with the onboard crew to ensure the capability to recognize and respond to inputs.

Data Bus/RAU's testing will require that the capability exists to issue commands to the RAU and read back data received at RAU.

The combination of the above tests will provide the capability to ensure the operational status of the subsystem data management system.

For test and checkout of the subsystem, the software will function in a control and monitoring mode and will perform such functions as limit testing, trend analysis, and discrete/analog input monitoring. Any anomalies encountered will result in: (1) Display on Subsystem CRT; (2) Telemetry of Associated Data; (3) Issuance of Caution and Warning Indication.

- Experiment Ground Checkout Set - The experiment ground checkout set will contain the software to perform test and checkout on the

experiment data management system and will perform those experiment interface tests not unique to the experiment hardware.

The test and checkout of the experiment data management system will result in tests analogous to those described previously for the subsystem software. The experiment interface test software will be utilized for diagnostic troubleshooting during integration testing.

#### EGSE Ground Checkout Set

To support the test and checkout of the Spacelab during its integration phases, electrical ground support equipment (EGSE) will be provided. This EGSE will consist of the computer, peripherals, simulators and software needed to support all levels of integration testing. The software will be contained within the EGSE ground checkout set which will provide the following capabilities: (1) Man/Machine Interface for Test Control and Monitor; (2) EGSE/CDMS Testing Control; (3) Stimulus Generation; (4) Data Recording/Data Reduction.

- Man/Machine Interface - The man/machine interface will be provided through the CRT/Console associated with the EGSE. The test conductor will be afforded the capability of selecting test options and controlling the test via the console. The man/machine interface capability will vary depending on the level of testing required. For detailed testing of hardware, significant interaction will be required; whereas, functional testing will require minimum interaction.

The following are examples of the capabilities which should be provided to the test conductor:

- Select test options
- Specify test sequences
- Specify test parameters/limits
- Specify stimulus requirements
- Specify data recording requirements
- Start/stop/repeat of testing
- Display of selected parameters
- Monitor parameters continuously

- Monitor parameters by exception
- Issue stimuli

In addition to the selection of predefined test software, the test conductor should be provided with a language to allow generation of basic on-line programs.

The man/machine interface will provide the test conductor with the ability to assess test status in a realtime environment and will allow rapid identification and isolation of problems. Within the extremely tight testing schedules currently planned (particularly at the launch site), the man/machine interaction is a required capability.

- EGSE/CDMS Testing Control - The EGSE software will maintain active control over the tests being conducted. To provide this capability, the EGSE software will accept input from the test conductor, format the commands, issue the commands to the CDMS software, verify response to commands, and communicate with the CDMS software, as required, during the tests.
- Stimulus Generation - To provide predictable input into the Spacelab subsystems and experiments, the EGSE software will control the stimuli. The ability to control the CDMS inputs while analyzing CDMS software actions in response to the inputs will provide an overall closed loop system test capability.
- Data Recording/Data Reduction - To support the integration testing levels, the EGSE software will provide data recording/data reduction capability. This capability will provide the means of performing detailed analysis of test data as required either during a test or in a post-test environment.

A functional representation of the EGSE capabilities in Spacelab testing is illustrated in Figure 3-6.

### 3.3.3.2 On-Orbit Test and Checkout Capability

The on-orbit test and checkout capability will be provided through the software existing within the CDMS flight set. As is shown in Figure 3-7, this set is comprised of the experiment flight set and the subsystem flight set.

#### Experiment Flight Set

The experiment flight set will contain software to perform test and checkout of the experiment data management system (similar to ground

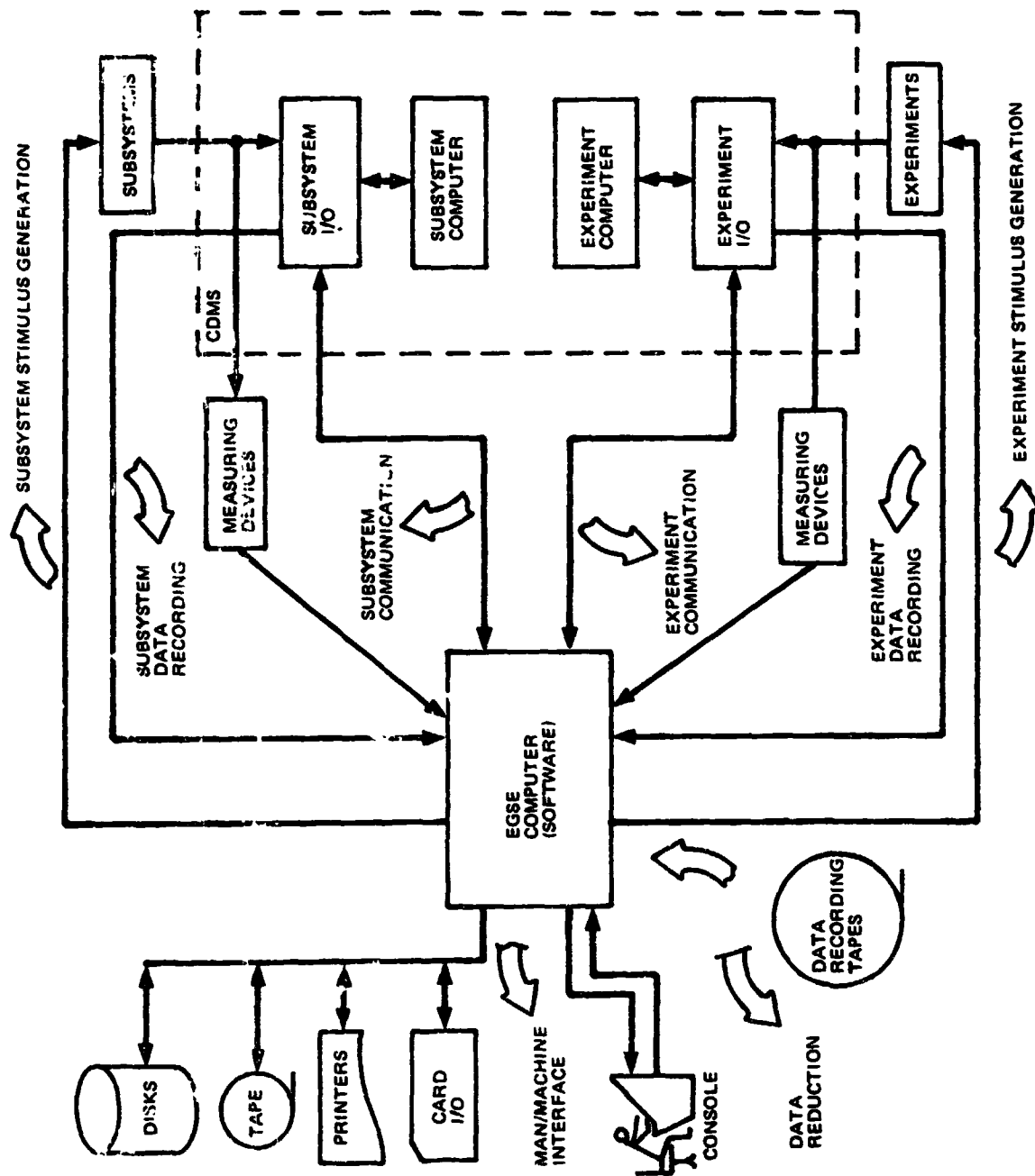


Figure 3-6. EGSE Software Role in Spacelab Test and Checkout

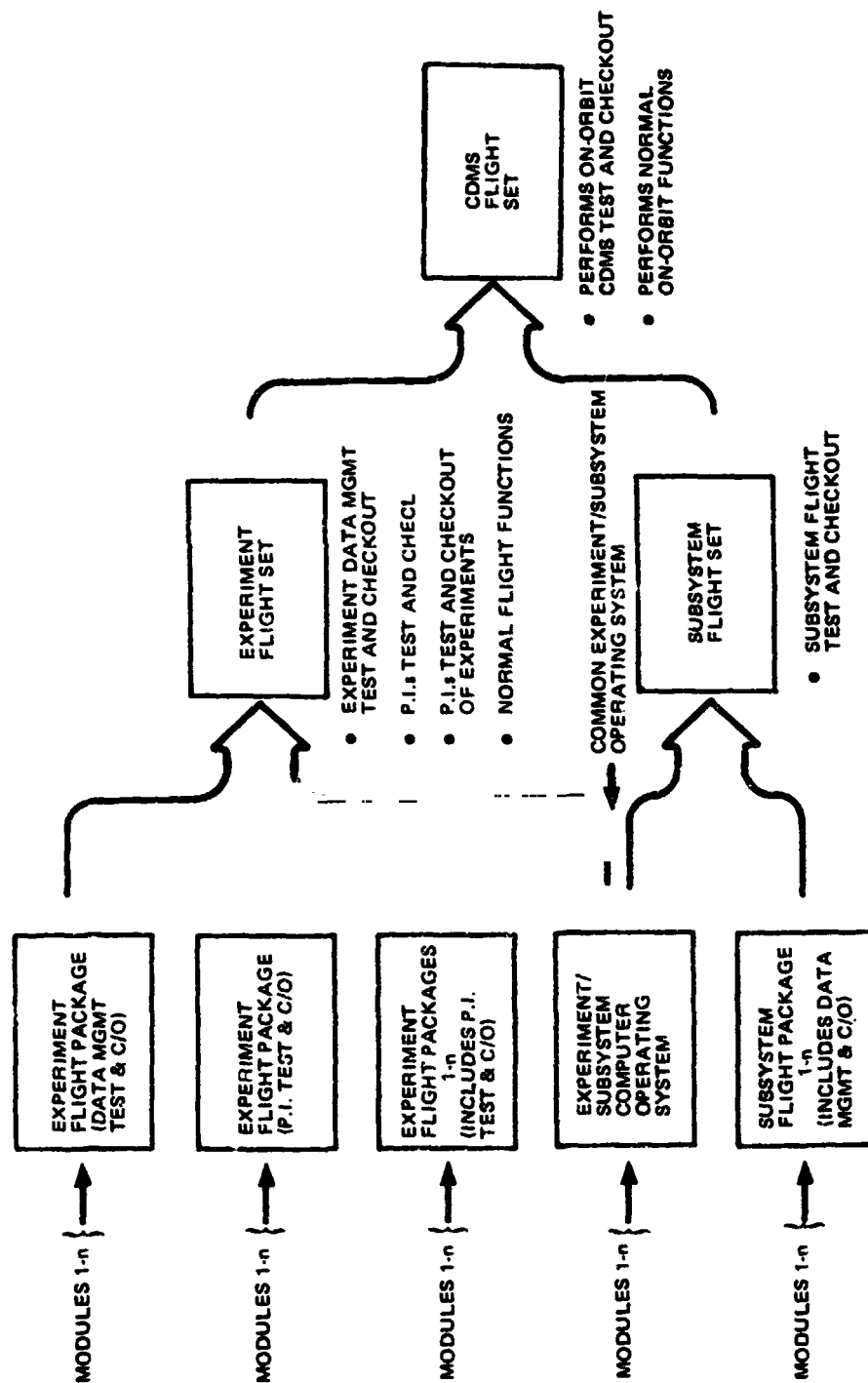


Figure 3-7. On-Orbit Test and Checkout Software



checkout capability) and will contain the software generated by the PI to perform test and checkout of the experiment hardware. The test and checkout software will be executed under control of the CDMS operating system in conjunction with the normal on-orbit processing functions required of the Experiment Flight Applications software.

#### Subsystem Flight Set

The subsystem flight set will contain the software required to perform subsystem data management system test and checkout and to monitor the performance of subsystems. Because the on-orbit subsystem software will only monitor the subsystems, actions to be taken as a result of anomalies will be under manual control or under remote control from ground or orbiter.

#### 3.3.3.3 Test and Checkout Software Utilization

The discussion in Paragraph 3.3.3.2 has identified the capabilities of the ground test and checkout software and the on-orbit test and checkout software. Within this section, the application of these capabilities to satisfy the Spacelab testing requirements will be discussed.

#### Ground Operations Test and Checkout

Within the Spacelab ground operations plan, three levels of hardware integration testing must be supported by the ground checkout software. These levels of integration testing facilities required, and the associated test and checkout software utilization are discussed in the following paragraphs.

- Level III Integration Testing - Level III integration testing will be performed at the Central Integration Site (CIS) and will consist of experiment/experiment module/pallet integration and testing and support section post-refurbishment integration and testing. The validation of the Experiment Flight Applications software will be included within the Level III integration testing.

To perform experiment/experiment module/pallet integration, the simulator of the Spacelab support section and the CDMS subsystem, provided by ESRO, will be utilized. The EGSE will also include stimulus generation equipment and measuring devices to record signals not available on the telemetry stream.

The experiment computer will contain either the experiment flight set for validation testing or the experiment ground checkout set for hardware integration testing. The contents of experiment

computer memory will be under control of the test conductor via the EGSE console. The onboard mass memory unit will be utilized for loading of the experiment computer with the required experiment software set.

The EGSE software will control the testing sequence, communicate with the experiment computer software set, record data, provide stimulus, and display test status and results to the test conductor. Under control of the test conductor, the capability to test all levels of integration from one experiment interface to total experiment interface complement will be provided.

The total software involvement in Level III integration of the experiment section is shown in Figure 3-8.

For this study, the assumption has been made that modification of the support section would require return to the CIS from the launch site. As can be seen in Figure 3-9, Level III integration of the support section will require use of ESRO provided hardware simulations of the orbiter interface and experiment segment/pallet. The orbiter interface simulator will allow testing of all interfaces between the support section and the orbiter; whereas, the experiment segment/pallet simulator will provide stimulus to the experiment and subsystem data buses.

The CDMS computer will contain either ground checkout or flight sets loaded from the mass memory, and the EGSE software will control test under direction of the test conductor. The test will consist largely of an interface test of the support section with the experiment segment/pallet simulator providing input to the support section and the EGSE software recording outputs of the orbiter interface simulator. The EGSE software will issue commands to the CDMS software via the uplink and will require responses via the telemetry system.

- Level II Integration Testing - Level II integration will consist of the integration of the Spacelab experiment section/pallets with the support section to create the total Spacelab system. This integration will occur at KSC and will require an overall system test of the Spacelab.

The CDMS computers will contain the ground checkout of flight sets, as required, to support the integration testing. Ground checkout sets will be used only if detailed testing is required. The EGSE will include stimulus generators and special recording devices along with the EGSE computer and associated software. As can be seen in Figure 3-10, the only simulator required will be the orbiter interface adapter.

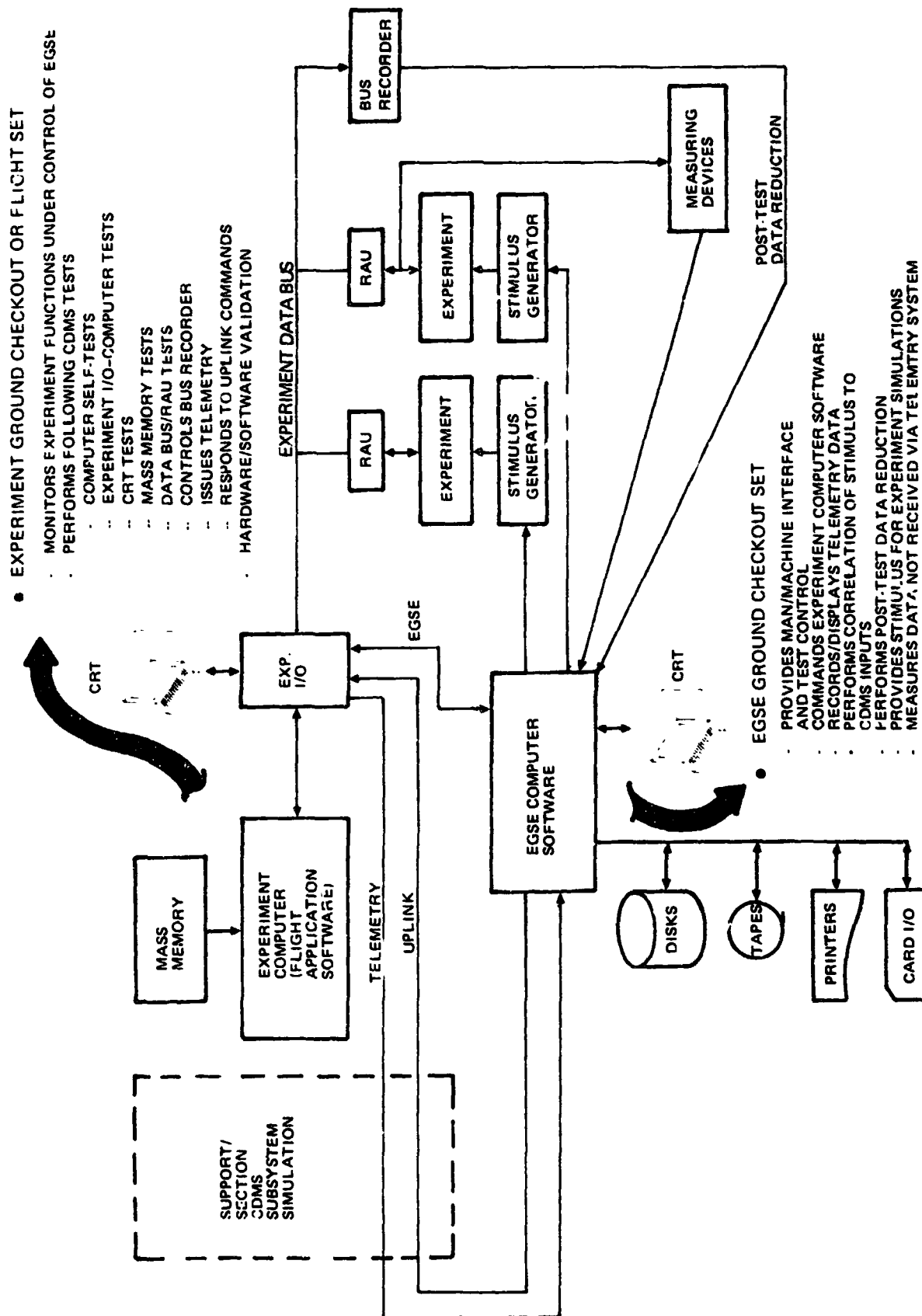


Figure 3-8. CIS Level III Testing Configuration

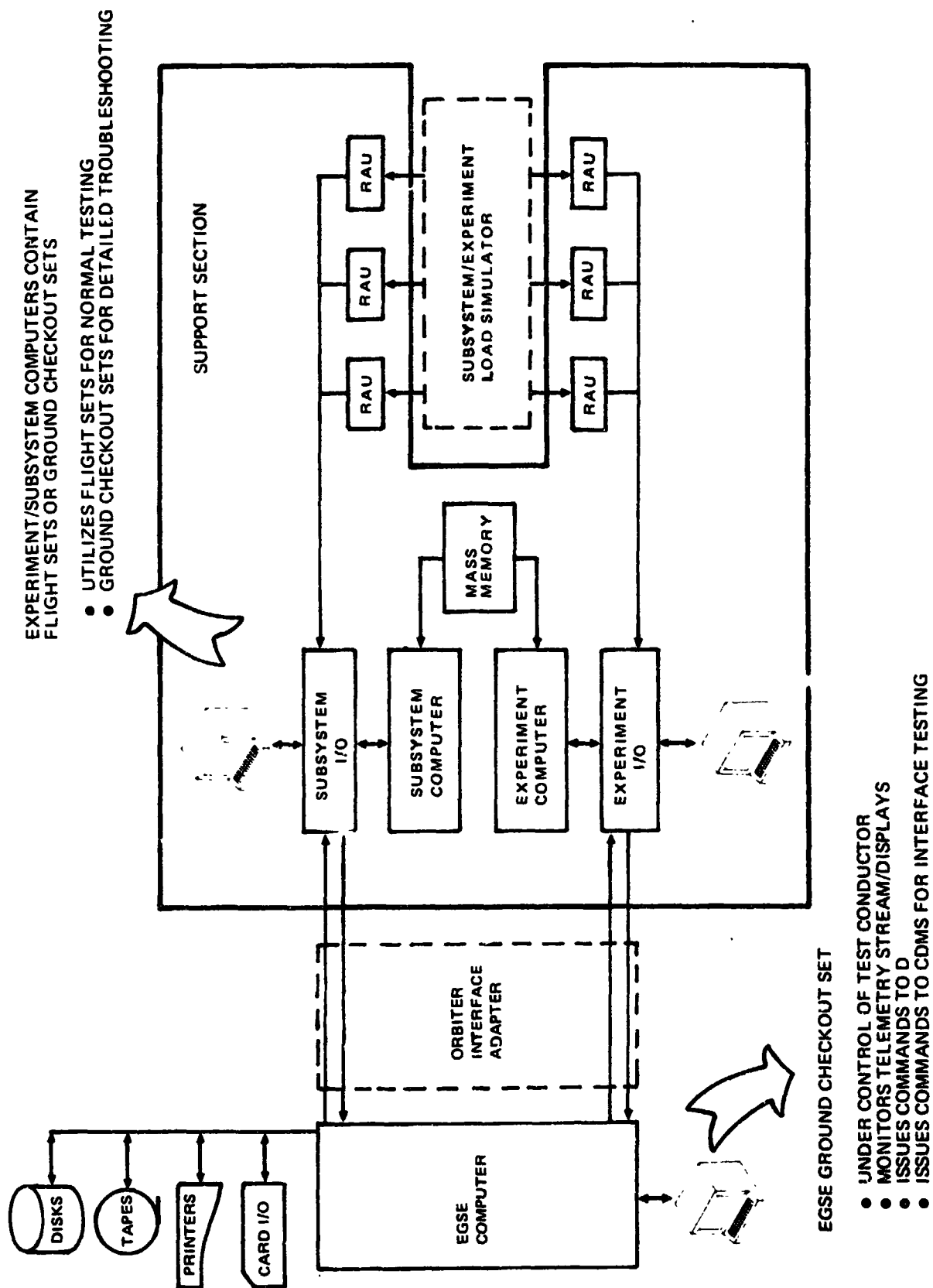


Figure 3-9. CIS Level III Testing Refurbishment of Support Section

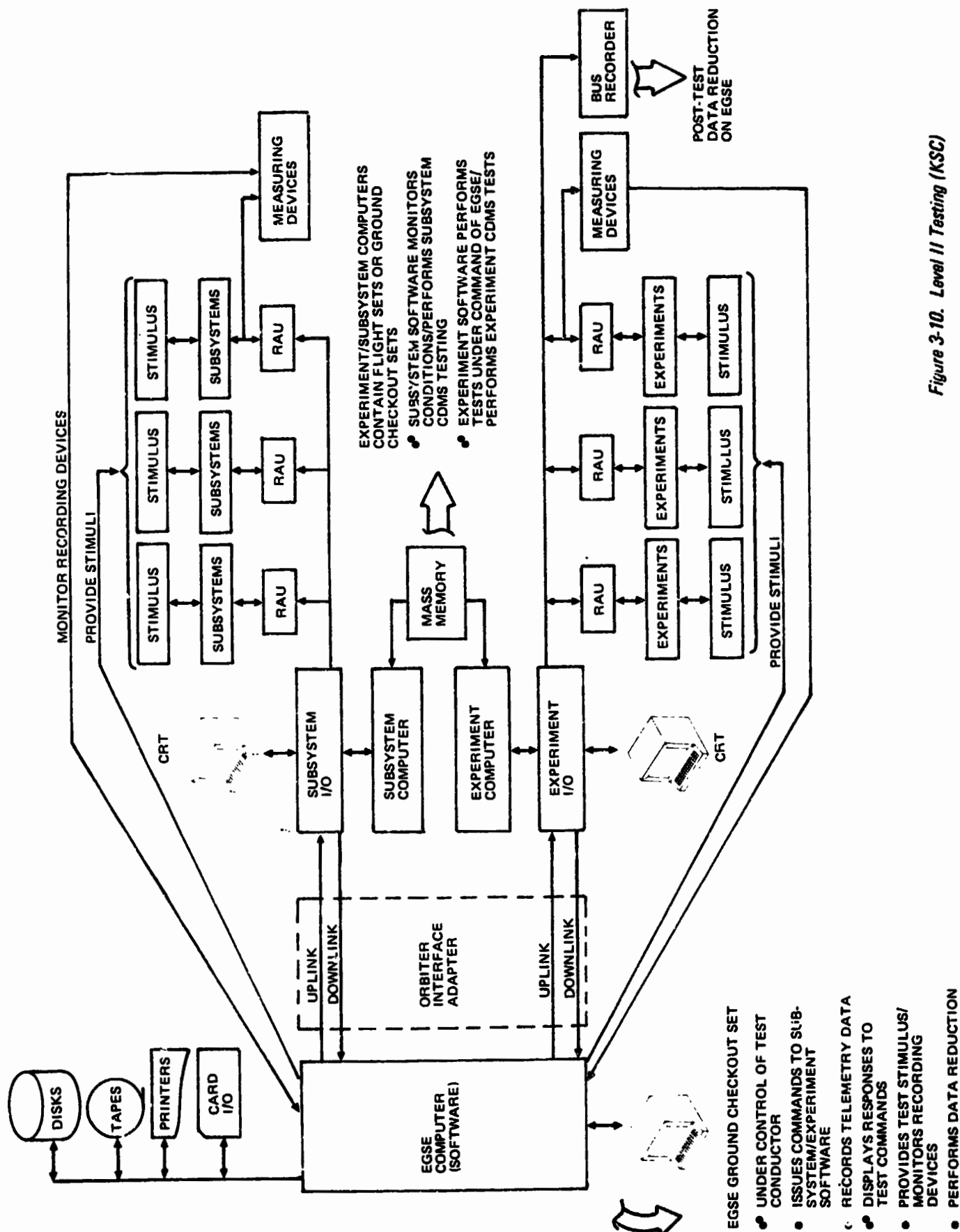


Figure 3-10. Level II Testing (KSC)

Because of the time constraints on the Level II integration testing, the tests will be of a functional nature. The stimulus generators will provide normal input data. Any significant problems will require launch delays or cancellations. Failures of individual experiment hardware may result in decision to launch with degraded mission objectives or the substitution of a backup payload.

The EGSE software will control the test sequence under control of the test conductor. Data reduction/analysis will be restricted to reduce the data volume and will only address overall Spacelab operational parameters.

- Level I Integration - Testing the Level I integration will consist of the integration of the Spacelab with the Shuttle Orbiter. This testing level will consist of tests controlled by the Space Shuttle Launch Processing System (LPS) and will ensure compatibility between Shuttle and Spacelab. The CDMS experiment and subsystem computers will contain the flight sets. The Level I integration utilization of software is shown in Figure 3-11. These tests will exercise the following interfaces:

- Caution and warning
- Telemetry downlink (low rate and high rate data)
- Command uplink
- PSS/Spacelab

Associated with the Level I integration will be an overall communication link test with the Payload Operations Center. This test will ensure compatibility of all elements associated with Spacelab on-orbit operations.

#### 3.3.3.4 On-Orbit Test and Checkout Utilization

On-orbit test and checkout will be performed by the experiment flight applications and subsystem flight sets. The subsystem software will monitor the operation of the subsystems, perform data management system tests on a periodic basis, and display the system status on the CRT. If anomalies occur, the associated data will be displayed, transmitted to the ground via the telemetry link, and if appropriate, a backup caution and warning will be issued.

The flight applications software will perform on-orbit monitoring and control of experiment hardware. If anomalies occur, the software will take action to by-pass the problem and/or notify crew/ground via display or telemetry. The software will also periodically perform data management system tests.

During on-orbit operations, the Payload Operations Center (POC) will monitor the status of Spacelab through telemetry stream analysis and will provide a display status. It has been assumed that the POC will not normally command the CDMS software to perform additional tests; however, through loading from the mass memory, such capability could be provided.

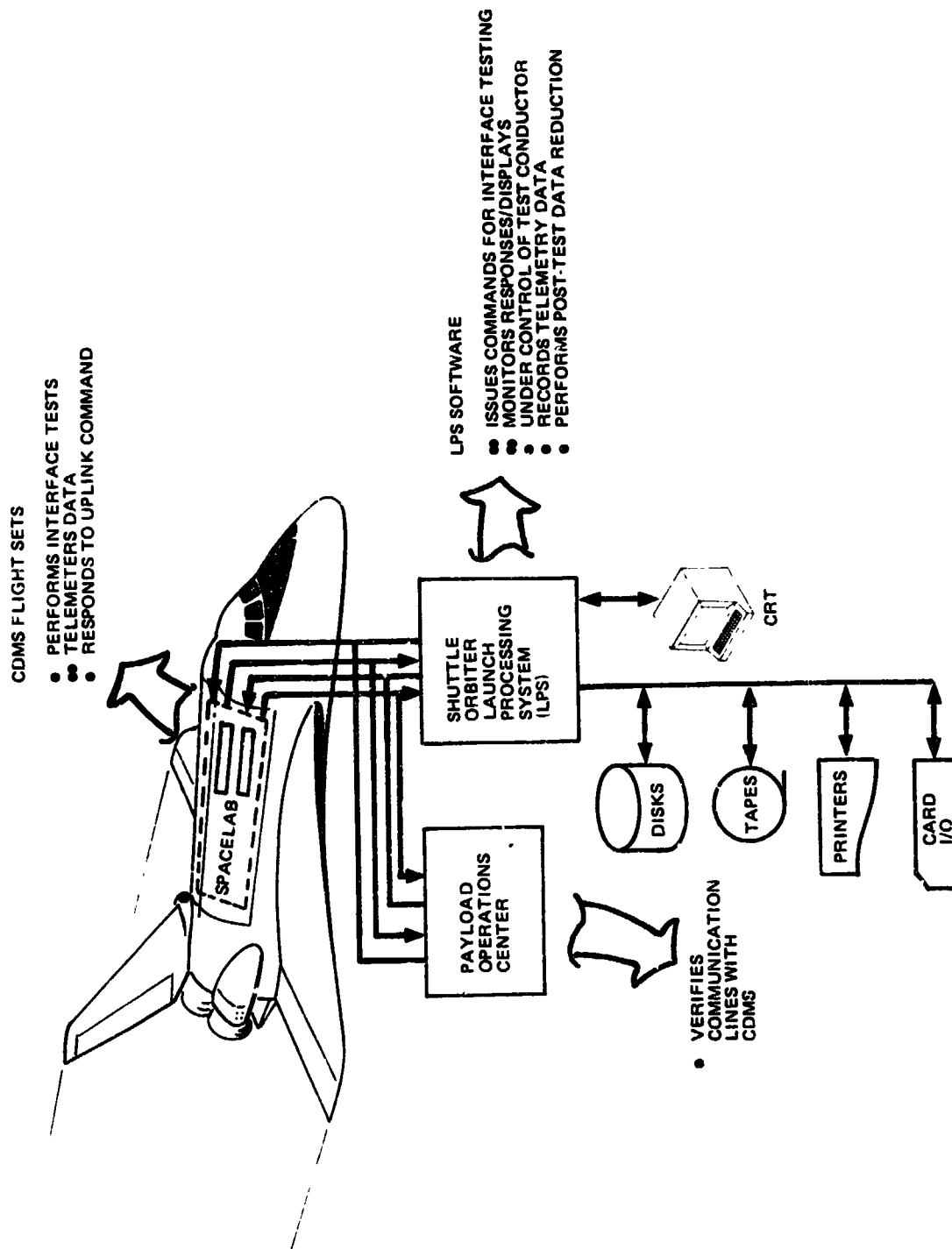


Figure 3-11. Level I Testing (KSC)

### 3.4 TEST AND CHECKOUT SOFTWARE DESIGN CONSIDERATIONS

#### 3.4.1 THEME

To support the wide range of Spacelab testing requirements, detailed testing to highly functional testing, the Test and Checkout software design philosophy must provide a maximum of flexibility while at the same time it must utilize common software, whenever possible, to reduce costs.

#### 3.4.2 CONCLUSION

The following design considerations have been established during this analysis:

- A common operating system should exist in both CDMS computers.
- A modular design concept should be utilized which will allow overlay of programs within the EGSE and CDMS computers.
- The EGSE and CDMS software designs must support man/machine interface.
- A Test and Checkout Language (GOAL-type) will be required for EGSE.
- An operating system should be provided for the EGSE.
- Data reduction/data analysis software within the EGSE should provide flexible redefinition of input formats, conversions, and output requirements.

#### 3.4.3 DISCUSSION

The CDMS software and EGSE software will provide the capability to perform test and checkout functions. The following paragraphs will discuss the design factors which should be considered during the development of test and checkout software.

##### EGSE Test and Checkout Software

To support the overall Spacelab goal of minimizing the use of EGSE during the testing phases, the EGSE software should be designed to provide the maximum possible testing flexibility. This would provide coverage over a wide range of testing requirements while maintaining a low change activity. To provide this flexibility, certain considerations must be included within the software design philosophy. These considerations are discussed in the following paragraphs.



- Operating System - To support the many testing configurations and corresponding software configurations, an operating system should be provided for the EGSE software. This operating system will provide a man/machine interface with the test conductor and will control the resulting software functions. A common operating system, thus, will provide a relatively stable environment for the test conductor throughout the overall testing environment.

The operating system will also provide a common interface medium for all application packages, and thus reduce interface problems and will enforce programming and standards on application programmers.

The EGSE operating system should provide the test conductor with the capability to generate test sequences, for support of test and checkout, through his test console. A procedure-oriented high-order language (such as GOAL) will provide this capability and must be considered within the operating system design.

- Application Packages - Because the experiment hardware will vary from mission to mission, it will be necessary to modify the EGSE software to support new hardware test requirements. The impact of this modification should be minimized through design considerations. To accommodate this environment, the application packages for the EGSE ground checkout set should be structured such that they are independent of each other and are executed under control of the EGSE operating system. This independence will allow the operating system to overlay existing EGSE computer memory with the appropriate package requested by the test conductor. In addition, this structure will allow separation of software applications such that software which is most subject to change can be handled in an individual manner and combined with unchanging software to create ground checkout sets for the EGSE.

An additional consideration impacting EGSE software design will be the required changes to measurement lists, test point definition, stimulus generation requirements, sensor calibration curves and test limits as the Spacelab hardware is modified. The software design should provide the capability to allow these parameters to be updated via tables, on a mission basis, and thus minimize testing time requirements on EGSE software. This capability could be expanded to build a parameter input tape for use at each testing facility for loading prior to a test. As a result, a standard EGSE baseline ground checkout set could be utilized without redelivery from mission to mission. Redelivery would be required only when EGSE software problems were encountered or when additional requirements are generated for the EGSE software.

The data reduction/data recording/data analysis software within the EGSE should be designed to allow rapid reconfiguration of input formats, conversion constants, and output formats through the use of test conductor inputs from the test console. This flexibility will allow continued utilization of the data reduction software with minimum modification throughout the Spacelab program.

#### CDMS Test and Checkout Software

As was discussed previously, the CDMS test and checkout software will consist of ground checkout sets and flight sets. Because common computers are utilized for both the ground checkout and on-orbit checkout functions, commonality of software across both testing regimes would appear feasible. To achieve this commonality, the design philosophy should attempt to encompass the test and checkout requirements and flight applications requirements whenever possible.

- CDMS Operating System - The operating system concept, discussed in Section 2 of this report, for utilization in development of Experiment Flight Applications software should be utilized for both the experiment and subsystem computer ground checkout and flight utilization. This common base will reduce development cost and will ensure common programming practices within both software development activities.
- Applications - The capability to perform overlays during both the ground test and checkout operations and the on-orbit test and checkout operations will necessitate a modular software structure in which applications are independent of each other.

An additional common element in both the experiment and subsystem test and checkout software is the self-test operations associated with the data management systems. Use of this common element will reduce development cost and ensure that the CDMS test and checkout is conducted identically within both systems.

A functional representation of one overall CDMS software design approach is shown in Figure 3-12.

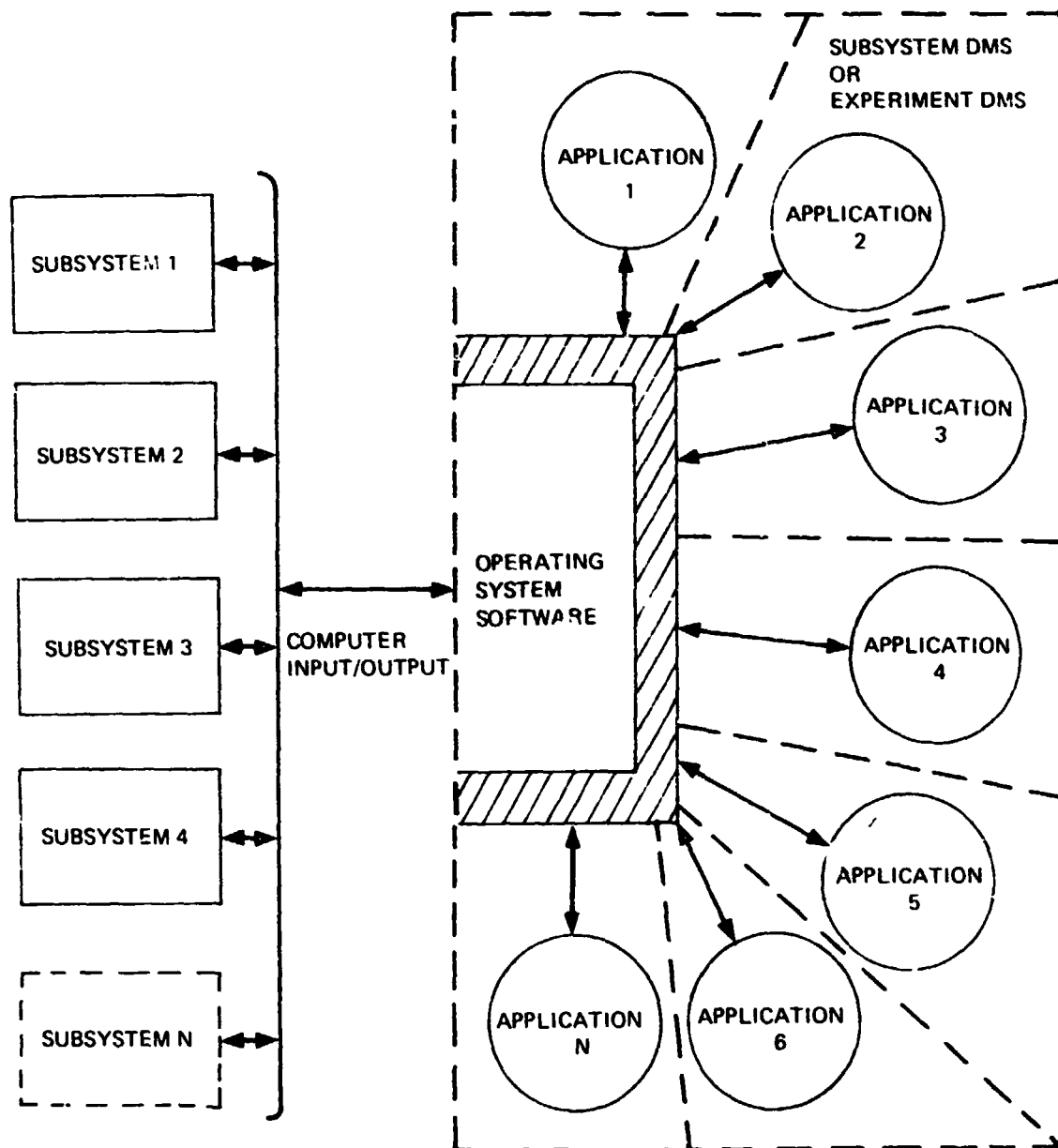


Figure 3-12. Functional Representation of CDMS Software Design Approach

### 3.5 TEST AND CHECKOUT SOFTWARE MAINTENANCE REQUIREMENTS

#### 3.5.1 THEME

To provide an estimate of the activities NASA must perform in maintenance of Test and Checkout software, an analysis of the size of the software and the anticipated change activity was performed. The associated software development tools and facilities required in the maintenance activity were also identified.

#### 3.5.2 CONCLUSIONS

The following conclusions regarding Test and Checkout software maintenance have been established during this study:

- NASA must maintain the personnel to support approximately 610 software modules associated with Test and Checkout software.
- Change activity of 2% to 5% per flight is anticipated for Test and Checkout software.
- The NASA STIL will provide the facility for Test and Checkout software maintenance.
- ESRO-developed support software packages will serve as the baseline for STIL support software; however, NASA must develop support software unique to the STIL environment.
- The major area of change within the Test and Checkout software will be that portion of the EGSE software required to support ground testing of the experiment flight applications software.

#### 3.5.3 DISCUSSION

The Spacelab Test and Checkout software developed by ESRO for the engineering model, flight unit 1, and flight unit 2, will provide the maintenance baselines for NASA. To support and maintain these software sets, NASA must have personnel familiar with the detailed operations of the software and must have the necessary software development tools and facilities for the maintenance function.

The following paragraphs will first address the magnitude of the Test and Checkout software and its anticipated change activity and will then address the required development tools and facilities.

#### Test and Checkout Maintenance Analysis

To establish the magnitude of the Test and Checkout software to be

maintained by NASA, a sizing analysis was performed. The results of this analysis are shown in Table 3.2.

As can be seen in the table, the Test and Checkout software to be maintained will consist of 610 modules. The capability to support and maintain such a large software base will require a significant number of programmers. In addition, a significant data base with software management tools will be required to maintain configuration control over all modules.

Analysis of previous experience on Test and Checkout software for both Saturn and Skylab programs indicates that the change activity within a maintenance environment will vary between 2% and 5% for each delivery. Such a change activity for Spacelab Test and Checkout software would require changing from 5700 to 14,250 instructions per delivery. For estimating the processing burden on maintenance facilities, it was assumed that the maximum change activity of 5% would exist.

The most constantly changing element of the Test and Checkout software will be the experiment interface portion of the EGSE ground checkout set. Since the requirements on this software element are a direct function of the experiment hardware, changes will be required for every flight which has experiment hardware which differs from the previous mission. Because of this ever-changing environment, the EGSE software will require redelivery for every flight.

Because of the anticipated stability of the Spacelab subsystems hardware, the CDMS Test and Checkout software should remain relatively stable throughout the Spacelab lifetime. The only changes will result from subsystem refurbishments, which modify the hardware to the extent that software must be updated, and software enhancements to improve test and checkout capability.

#### Development Tools

The development tools utilized for Test and Checkout software maintenance will consist of both ESRO-developed and NASA-developed tools. Each tool required will be discussed in the following paragraphs along with the origin of the capability. Many of the tools are identical to those previously discussed for Experiment Flight Applications software.

- Environment Models - The environment models must simulate the operating environment in which the Test and Checkout software must perform. The fidelity of these models will vary depending on the level of testing - low fidelity for functional level testing and high fidelity for verification. Models of the following have been identified for use in Test and Checkout software maintenance:

- Spacelab vehicle
- Shuttle Orbiter interface

**Table 3.2. Size of Test and Checkout Software to be Maintained by NASA**

TEST AND CHECKOUT SOFTWARE MAINTENANCE	SIZE (INSTRUCTIONS)	HIGH ORDER STATEMENTS *	NUMBER OF MODULES**
<b>CDMS SOFTWARE MAINTENANCE</b>			
– Subsystem Ground Checkout Set	55.1K	11K	110
– Subsystem Flight Set	14.23K	2.8K	28
– Operating System	20.0K	4.0K	40
– Experiment Data Management Checkout	2.5K	.5K	5
– Experiment Ground Checkout	16.7K	3.34K	33
<b>EGSE SOFTWARE MAINTENANCE</b>			
– Operating System	7.0K	1.4K	14
– EGSE Self Test	30.0K	6.0K	60
– Ground Checkout	26.2K	5.2K	52
– Experiment Interface	5.0K	1.0K	10
<b>SUPPORT SOFTWARE MAINTENANCE</b>			
– Simulators	101.0K	20.2K	202
– Data Reduction	28.0K	5.6K	56
<b>TOTALS</b>	<b>301.7K</b>	<b>61.0K</b>	<b>610</b>

\* High Order Language Statements determined by applying ratio of 5:1 to number of instructions.

\*\* Number of modules determined by assuming structured 100 HOL statements per module.

- Experiments
- Payload operations interface
- EGSE interface
- Spacelab subsystems

These environment models will be developed by NASA and will likely be those models developed for utilization in development of Experiment Flight Applications software (Section 2 of this report).

- CDMS/EGSE Interpretive Computer Simulations (ICS) - The ICS for both the CDMS and EGSE computers will be made available by ESRO. If the CDMS and EGSE computers are identical, only one ICS will be required; however, the present plans for test and checkout call for different computers. The ICS will provide the capability to perform detailed logic testing of both the CDMS and EGSE software but use will be minimized because of the excessive computer time required to perform testing.
- CDMS/EGSE High Order Language (HOL) - The CDMS and EGSE Test and Checkout software will be written in a HOL. Use of a HOL will be supported through use of compilers/assemblers/linkage editors made available to NASA by ESRO.
- CDMS Functional Simulators - The functional simulator provides the capability to simulate the HOL statements of the CDMS Test and Checkout software in the language of a host computer. This simulation will be developed by NASA.
- CDMS Subsystem Simulation Language - To support testing of the CDMS Test and Checkout software, a language will be provided to allow rapid development of models. This language will be procured or developed by NASA.
- Automated Configuration Management - The automated configuration management system must provide the capability to allow monitoring and control over the software change activity and configuration control over released software modules, packages, and sets. The system will be developed by NASA for its use in maintenance of Test and Checkout software.

### Development Facilities

It has been assumed that the Test and Checkout software will share development facilities with the Experiment Flight Applications software. As a result, the Software Test and Integration Laboratory (STIL) must have sufficient capability to support the maintenance of Test and Checkout software as well as supporting development of the Experiment Flight Applications software.



### 3.6 STIL REQUIREMENTS

#### 3.6.1 THEME

The maintenance and support of Test and Checkout software has been identified as a STIL responsibility; therefore, the utilization of computing resources required to support this activity imposes an impact on the STIL facility. This section assesses the impact of supporting software management, maintenance, integration, verification, and delivery on the STIL.

#### 3.6.2 CONCLUSIONS

Test and Checkout software maintenance and support will require 62 runs per day during operational phases.

#### 3.6.3 DISCUSSION

It has been established that the STIL must provide the maintenance and support for the ESRO-developed Test and Checkout software. This activity will be performed utilizing the tools and software characteristics discussed in Paragraph 3.5. The impact on the STIL created by the maintenance and support requirements must be determined for use as input to the STIL modeling task in Paragraph 5.4.

Since the Test and Checkout software must be maintained by NASA after delivery from ESRO, support functions such as software management, software maintenance and software integration must be provided.

Since there is experiment ground checkout software which is unique to each application, the overall ground checkout software sets must be developed in conjunction with the Experiment Flight Applications sets. The development activity in 1981 was the basis for analysis of flight application software development requirements; therefore, the same time period has been used for analysis of Test and Checkout software maintenance. As a result, nine test and checkout sets must be in process simultaneously to accommodate the delivery of EGSE/CDMS sets.

These considerations have been used in developing the estimated Test and Checkout software utilization of STIL. The following discussion is based on the aforementioned assumptions and analysis.

##### 3.6.3.1 STIL Requirements for Test and Checkout Software

In order for NASA to perform maintenance of Test and Checkout software, the following required functions have been identified:

- Software management
- Software maintenance
- Software integration

The discussions which follow will establish the load placed on STIL by the above functions. In establishing the loads, the approach taken was to determine the requirements of one package or set during a cycle. Once this load had been established, the multiple package/set factor, which represented the simultaneous activity during the cycle, was utilized to generate a total load for the cycle. Since the STIL modeling exercise required loads to be established on a daily basis, the total cycle loads were then divided by 30 days (6 months, 5-day week).

#### Software Management

Software management must provide the capabilities of tracking all change activity and generating software delivery data through the use of automated methods. These automated methods must be provided by the host facility. The requirements to process 9 EGSE/CDMS test and checkout sets within a 6 month period indicate a severe need for software management capabilities.

As with Experiment Flight Application software, there has been an assumption made of one run per day to maintain the Test and Checkout software data base. To remain responsive to Test and Checkout software needs, reports of problems and change activity must be supplied weekly.

Experience has indicated that for programs which have stabilized, one release/delivery should be sufficient. Therefore, for test and checkout, only one final release has been indicated in the load requirements.

The summary of STIL load resulting from software management is shown in Table 3.3.

#### Software Maintenance

The most resource consuming function related to support and maintenance of Test and Checkout software will be software maintenance. Activities relative to software maintenance are as follow: (1) Compiles/Assembles/Link Edits, (2) Interpretive Simulation, (3) Functional Simulation, and (4) Data Reduction.

In order to assess the impacts on STIL by test and checkout maintenance, the above activities will be analyzed individually.

Compiles/Assembles/Link Edits - Even though ESRO will deliver the Test and Checkout software to NASA, compiles/assembles/link edits will still be needed to update the software. Historically there has always been change activity with any ongoing programs, and it is anticipated that Spacelab will be no different in that respect. Using the bases established earlier of 610 modules, 5% change activity per package and assuming 9 compiles/assembles/links per change, the following formula represents the anticipated run load per package/set:

**Table 3.3. Summary of Test and Checkout Impact on STIL**

FUNCTION	NEW FLIGHT OR REFLIGHT RUNS PER		
	PACKAGE/SET	CYCLE	DAY
<b>SOFTWARE MANAGEMENT</b>			
– Configuration Management and Statistics	130 26	1170 234	9 1.8
– Automated Release	.1	9	.1
<b>SOFTWARE MAINTENANCE</b>			
– Compiles/Assembles/Links	274	2466	18
– Interpretive Simulation	137	1233	9
– Functional Simulation	137	1233	9
– Data Reduction	205	1845	14
<b>SOFTWARE INTEGRATION</b>			
– System Set Build	2	18	.14
– System Set Integrated Test	2	18	.14
<b>TOTAL RUNS PER DAY = 62</b>			

$(610 * 5\%) * 9 \text{ Compiles/Assembles/Links} = 274 \text{ Total Runs per Package}$

The 274 total runs per package can be rounded to approximately two runs per day per package. For all (9) test and checkout packages in process at any time in the cycle, there will be approximately 18 runs per day.

- Interpretive and Functional Simulation - To assure the quality of changes made to Test and Checkout software, simulation runs will be required. By the time NASA assumes total responsibility for the Test and Checkout software, the changes should not entail significant logic revisions; therefore, the simulation activity has been assumed to require no more than an average of one run for each compile/assemble/link. This estimate yields a total of 18 simulation runs/day for (9) packages/sets in progress at any time.
- Data Reduction - For data reduction utilization, an estimate of 75% of the daily simulation load has been used. This estimate is based on the composite interpretive and functional simulation loads. Therefore, data reduction has been determined to required 14 runs per day.

The summary of STIL load requirements from software maintenance is shown in Table 3.3.

#### Software Integration

In order to ensure the integrity of each delivered Test and Checkout software set, there must be an integrated test. Once a Test and Checkout software set has been manufactured in the host facility, an integrated test will then be performed. Since IS schedules will be so restrictive, the STIL will be required to support an extensive integrated test prior to shipment to CIS.

For software integration, the assumption was made that 2 runs would be required for system set build as well as system set integration. Table 3.3 reflects these requirements.

#### 3.6.3.2 Summary of Test and Checkout Impact on STIL

Table 3.3 reflects the total load placed on STIL by Test and Checkout software. The daily loads have been shown as fractional loads; however, the final total has been represented in rounded form.

## TASK 4B: MISSION OPERATIONS

### 4

SECTION		PAGE
4	TASK 4B: MISSION OPERATIONS . . . . .	4-1
4.1	TASK 4B: SUMMARY . . . . .	4-1
4.1.1	Objectives/Study Approach . . . . .	4-1
4.2	MISSION SUPPORT SYSTEM DESCRIPTION . . . . .	4-3
4.2.1	Theme . . . . .	4-3
4.2.2	Conclusions . . . . .	4-3
4.2.3	Discussion . . . . .	4-3
4.3	DATA FLOW ANALYSIS . . . . .	4-9
4.3.1	Theme . . . . .	4-9
4.3.2	Conclusions . . . . .	4-9
4.3.3	Discussion . . . . .	4-9
4.4	MISSION PLANNING SYSTEM (MPS) . . . . .	4-11
4.4.1	Theme . . . . .	4-11
4.4.2	Conclusions . . . . .	4-11
4.4.3	Discussion . . . . .	4-11
4.4.3.1	MPS Description . . . . .	4-13
4.4.3.2	MPS Utilization . . . . .	4-15
4.5	PI PARTICIPATION IN MISSION OPERATIONS CONCEPT . . . . .	4-17
4.5.1	Theme . . . . .	4-17
4.5.2	Conclusions . . . . .	4-17
4.5.3	Discussion . . . . .	4-17
4.5.3.1	Software Requirements . . . . .	4-17
4.5.3.2	Spacelab PI Interface Concepts for Mission Support . . . . .	4-18
4.6	STIL SUPPORT REQUIREMENTS . . . . .	4-21

## TASK 4B: MISSION OPERATIONS 4

### 4.1 TASK 4B: SUMMARY

The operational environment to be supported by the mission operations function of Spacelab is much more severe than the environment of previous scientific missions. This severity is directly attributable to the following factors:

- Each new mission may contain significantly different scientific payloads requiring different mission operations support.
- Data rates, data preprocessing, and data storage procedures are orders of magnitude greater than previous space programs.
- Active participation of PIs in realtime scientific experiment monitor and control must be supported.
- Rapid turnaround requirements on experiment data is required before next Spacelab mission on which same experiment will be flown.
- Mission planning for experiment operation must support 12-day launch cycles of Spacelab.

These Spacelab considerations will necessitate new concepts for mission operations. This report section will discuss the requirements which must be satisfied by the mission operations concepts to be utilized for Spacelab.

#### 4.1.1 OBJECTIVES/STUDY APPROACH

The objective of the mission operations study task was to identify those mission operation functions which have impact on the capability of the Spacelab to achieve its scientific objectives. The primary areas of analysis were:

- Mission support system definition
- Data flow
- Mission planning system
- PI participation in mission support
- STIL support requirements

The conclusions reached as a result of the above analyses establish a mission operating concept for Spacelab.

## 4.2 MISSION SUPPORT SYSTEM DEFINITION

### 4.2.1 THEME

The Spacelab CDMS, the Communication Network, the Shuttle Orbiter, the Payload Operations Center (POC), and the Preprocessing Facility are the elements of the mission support system for Spacelab. The combination of the capabilities of these elements satisfies the mission operating support requirements.

### 4.2.2 CONCLUSIONS

The following conclusions have been reached regarding the mission support system:

- NASA will provide a preprocessing facility for data reduction prior to dissemination to user.
- The Shuttle Orbiter will provide the communication link between the ground and Spacelab.
- The CDMS software will support the mission operations interface to the experiments.
- Tracking and Data Relay Satellite System (TDRSS) will provide the primary downlink/uplink capability.
- The Payload Operations Center must provide a PI interface.

### 4.2.3 DISCUSSION

The interactions among the identified Spacelab elements provide the mission operations concept for Spacelab.

#### Spacelab CDMS

The Spacelab CDMS will contain the Experiment Flight Applications (EFA) software which provides the direct interface with the experiment hardware. Through this interface, control and monitoring of experiment operation is provided. This software will provide the capability to interface with a PI either from the ground via the uplink or from the onboard experiment consoles. The software will be reconfigurable, under control of the PI, to provide the flexibility to address and resolve contingency situations. The EFA software will also provide the telemetry data, via the downlink, necessary for ground monitoring and control of experiments.

### Orbiter

The communications link between the Spacelab and the Space Transportation System (STS) communications network will be provided by the Orbiter. This is in keeping with Shuttle philosophy in support of payloads. The exact hardware configuration of the onboard transmission system is currently to be defined; however, several requirements to be supplied by the system have been identified. Science data in telemetry format must be transmitted intermittently at very high data rates; therefore, availability of TDRSS and the capability of maintaining an open KU band single access channel have been assumed. The attitude control and directional antenna pointing functions are assumed to be performed by the Orbiter in accordance with the pre-defined mission plan.

### Communications Network

A recent study performed by IBM under contract to MSFC (Contract No. NAS8-14000) determined that the most cost effective approach for transmitting data from the Spacelab to the ground was to use the TDRSS for total RF return. TDRSS was found to be the only telemetry system planned to be in operational status in the 1980-1990 time frame which will be capable of addressing the problem of scientific data telemetry rates of up to 50 MBPS.

Communications from a ground station to the Payload Operations Center may be accomplished by land lines grouped in increments of 1.344 MBPS to satisfy the mission/experiment specified data rate or by retransmission employing commercial communications satellites.

### Payload Operations Center (POC)

The Payload Operations Center (POC) will be located at one of the NASA centers and will support a single or a group of similar scientific disciplines. The POC will house the facilities, software, and personnel required to support the onboard activities. The critical elements within the POC are the mission operations computer system and associated software. A representative Payload Operations Center configuration is shown in Figure 4-1.

The POC software will consist of the following major elements:

- High-speed telemetry subsystem
- Mission control program
- Data evaluation
- Support processor
- Mission control executive
- Telecommunications processor



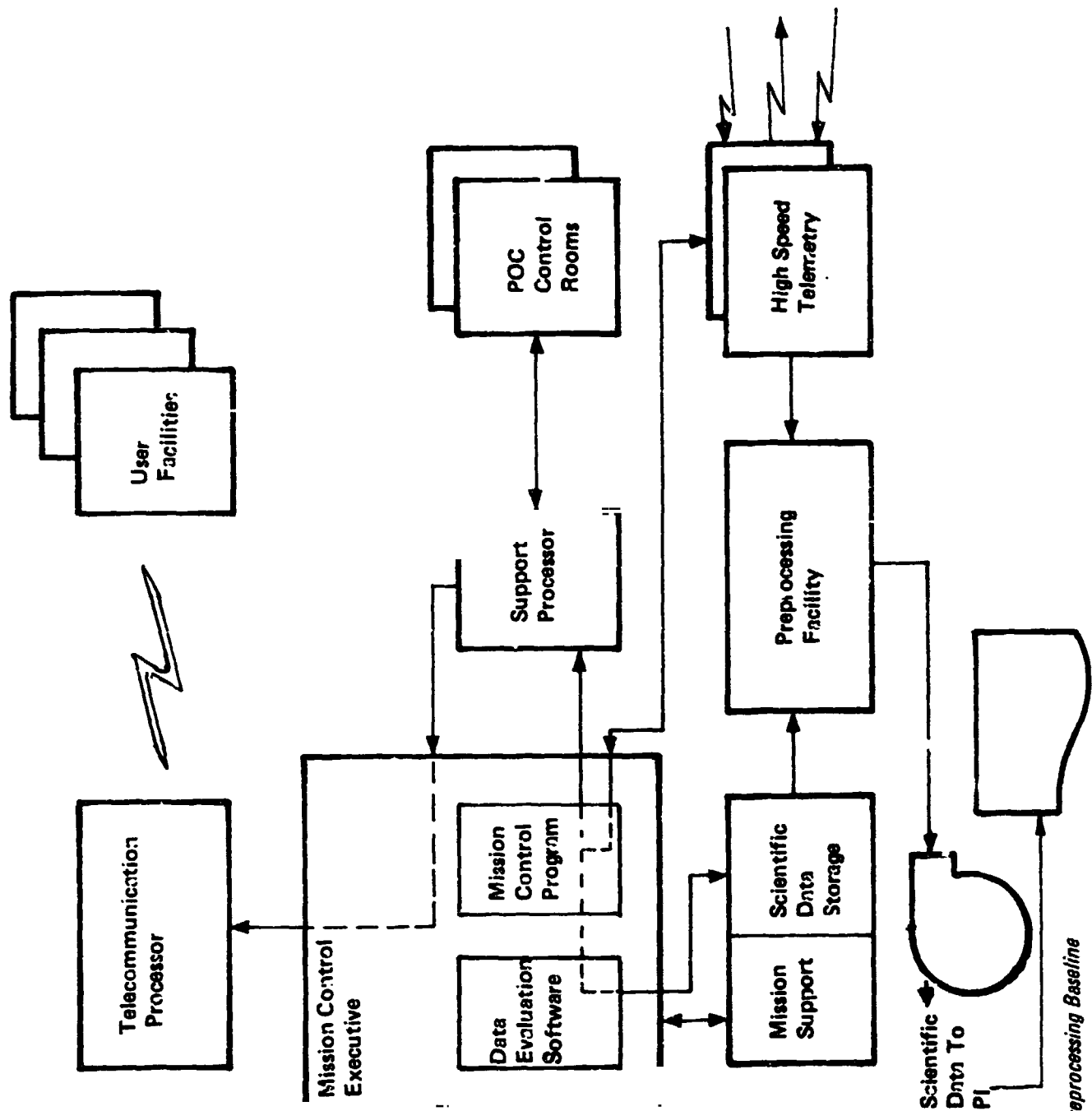


Figure 4-1. Spacelab POC & Preprocessing Baseline

The high-speed telemetry subsystem performs decommutation of telemetry data, determines the source of the data and identifies the type of data being received for the personnel within the control room. The decommutation process separates the data into predefined streams for display or storage operations.

The mission control program is responsible for performing a switching function to establish the data path between the telemetry system and the experimenter's console located within the POC. Data from the telemetry system is sent directly to a support processor where it is prepared for presentation on a graphics display device at the appropriate console within the control room.

The data evaluation software provides the PI with the capability of deleting or retaining portions or all information within a data base for more detailed evaluation and selective elimination in a non-realtime environment.

The support processor provides the capability of supplying realtime science data to remotely located consoles at user facilities. Functions performed at these remote locations could be identical with those that can be performed at the consoles located at the POC within the limitation of the speed of the communications link to the remote site.

The control of the Payload Operations Center (POC) complex is performed by the mission control executive (MCE) system residing in the primary computer under supervision of a realtime operation system. In addition to controlling the individual elements of the complex, the MCE is responsible for execution of a large variety of mission dependent application software which performs such functions as: (1) monitor Spacelab subsystems and payload health data, (2) perform command validation and uplink, (3) maintain updated trajectory and ephemeris information, (4) maintain a mission log, and (5) provide mission management information from the mission support data base upon request.

Because of the onboard checkout philosophy employed in Spacelab, the requirements for realtime monitoring of experiment operation will be less than previous space requirements. The POC software will monitor the downline data for failure indications telemetered by the CDMS computer; however, no detailed modeling of onboard systems or limit testing/trend analysis will be required. As a result, no dedicated flight controllers for Spacelab status monitoring will be required.

The telecommunications processor provides teleprocessing network control, message queuing and routing for a network of low speed remote terminals at other NASA centers and at selected user facilities. These terminals will primarily be used for obtaining mission status information and are not critical to the operation of the POC.

### Preprocessing Center (PPC)

The Preprocessing Center should be co-located with the POC in order to share a common data bank containing payload generated scientific data and thus reduce the lines of communications which would be required by a separately located preprocessing facility. NASA will provide the software utilized in preprocessing.

The PPC software will perform certain processing on the downlink data stream to provide output to the user in a form specified prior to the mission. These processes include annotation, calibration, conversion to engineering units, filtering, image processing, and quality testing.

The user of a PPC will relieve the PI of the processing burden associated with many of the Spacelab experiments and will allow PIs with small-scale computer facilities to participate in the Spacelab program.

#### 4.3 DATA FLOW ANALYSIS

##### 4.3.1 THEME

The flow of data from a sensor to the PI for evaluation of experiment results is an important function of the mission operations concept for Spacelab. The high data rates and volume of data impose requirements on the mission operations concept.

##### 4.3.2 CONCLUSIONS

- Preprocessing of data prior to turnover to the PI is a NASA requirement.
- Rapid dissemination and evaluation of data from frequently flown experiments is required for future mission planning.
- PI/crew interaction to reduce volume of data will be a goal.

##### 4.3.3 DISCUSSION

###### Spacelab Data Rates/Volume

In defining the data flow for Spacelab, an examination of the types of data and data rates was performed. The information was obtained from previous studies performed by IBM and indicated that data rates of up to 50 MBPS should be expected. These data rates will result in data volumes for some missions approaching  $5.4 \times 10^{11}$  bits. Skylab data rates, which were considerably lower than Spacelab's data rates, created data volumes which completely inundated the processing facilities allocated. Without proper planning, the problem of data volume will be unmanageable on Spacelab.

To avoid the problem of excessive volumes of data, a concept known as "interaction" is currently being studied. This concept requires that either visual or computerized techniques be utilized to evaluate the quality and/or the validity of data, and reject, from further processing, that data not considered useful. Although still in a preliminary phase of development, such an approach could be of great benefit on the Spacelab program.

###### Data Flow

The flow of the experiment data begins at the sensor and proceeds through the elements of the mission operations system to the PI. This flow is illustrated in Figure 4-2. The "interaction" activity previously described will be accomplished at the Spacelab CDMS consoles, Payload Operations Center, or Preprocessing Center. The output of the Preprocessing Center will be in a format acceptable to the user for further processing on his computer facilities.

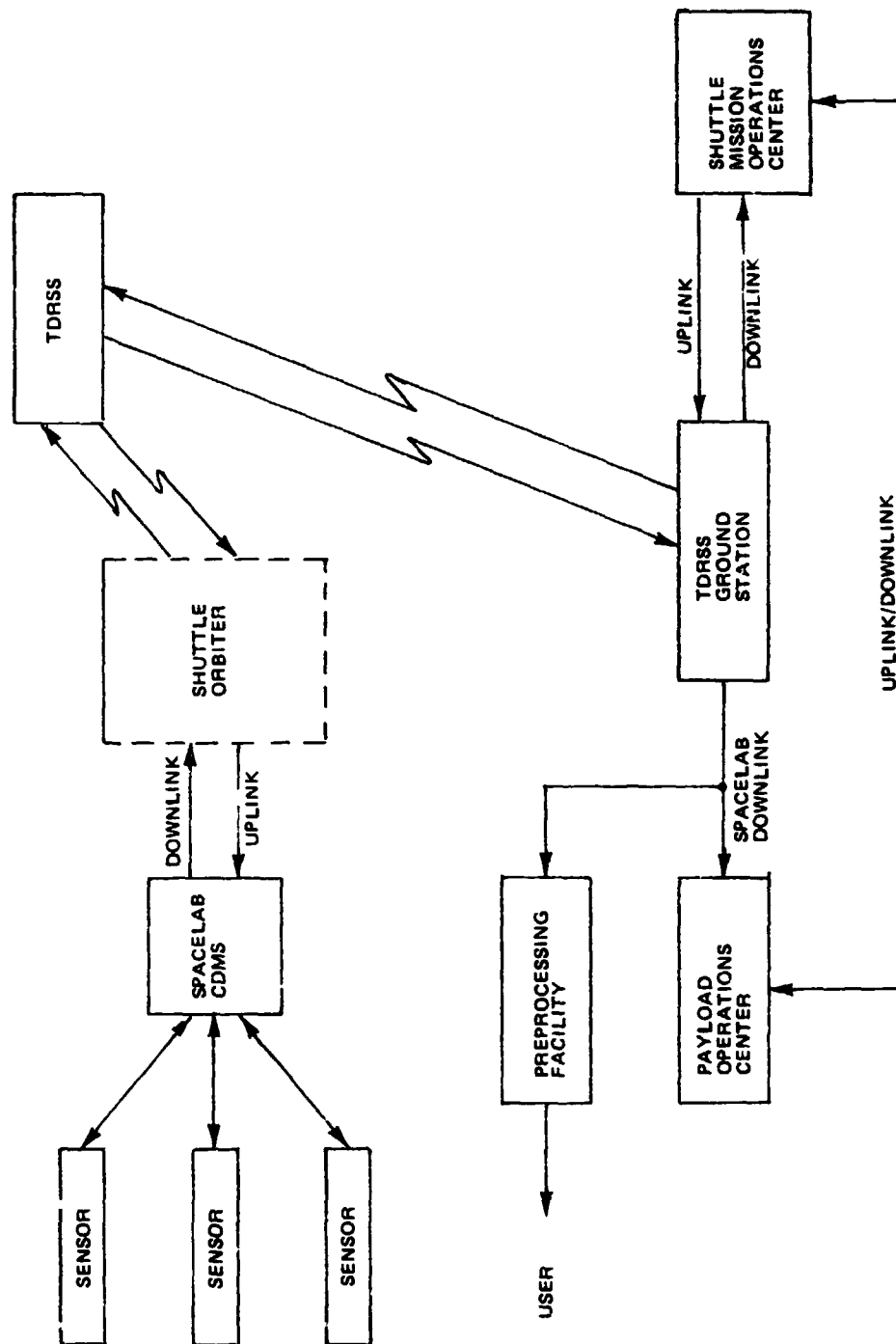


Figure 4-2. Spacelab Data Flow

#### 4.4 MISSION PLANNING SYSTEM (MPS)

##### 4.4.1 THEME

In an environment of rapid turnaround of Spacelab missions, the Mission Planning System is a critical element in achieving the required Spacelab mission objectives of:

- 31 flights/year maximum
- 12-day turnaround between missions
- Maximum scientific data coverage
- Optimum utilization of onboard crew

##### 4.4.2 CONCLUSIONS

The Spacelab Mission Planning System (MPS) must satisfy the following requirements:

- Provide pre-mission planning of experiment utilization/crew activity during mission.
- Be interactive with the planner in building mission timelines.
- Allow inflight modification of timelines for contingency operations.

##### 4.4.3 DISCUSSION

The primary purpose of the MPS is to provide an interactive tool for assisting mission planning personnel in the development of viable timelines and flight plans for Spacelab missions. Input to the system is a preliminary definition of the experiments which make up a payload and the associated Spacelab configuration. Using the facilities provided by the MPS, the mission planner can assure the operational compatibility of the elements which must interact to achieve the goals of the mission (for example, the experiments, the Spacelab and its subsystems, the orbiter, the communication network, and the ground support facilities). Output of the planning cycle is the predefined mission plan which is maintained on-line at the Spacelab POC where it can be modified and updated should contingency situations arise. A condensed copy of the plan may also be maintained onboard in the form of briefing material within the Spacelab CDMS where it can be easily referenced by the crew. Capability exists for generation of modified timelines in the event the original plan must be adjusted. Modified plans may be produced interactively using the MPS and loaded into the POC and Spacelab CDMS for implementation.

The mission planning process is functionally shown in Figure 4-3.

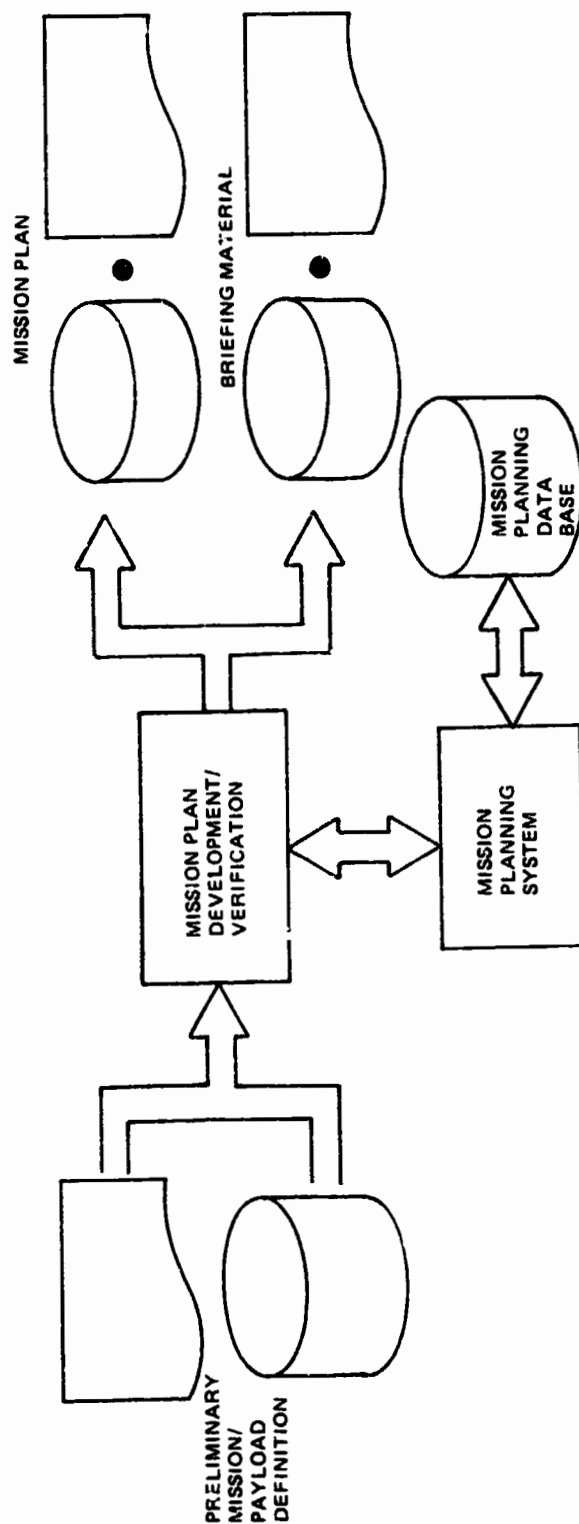


Figure 4-3. Mission Planning Process

#### 4.4.3.1 MPS Description

The MPS is envisioned as primarily an on-line, interactive software system available to users at remotely located terminals. Access to the system in a batch mode is provided for those planning activities which do not require immediate response, such as the introduction of the preliminary mission profile and the final verification of the total flight plan. A possible architecture for the MPS is shown in Figure 4-4. The software elements are the Executive, the Terminal Management Subsystem (TMS), the Display Formatter Subsystem (DFS), the Data Base Management Subsystem (DBMS), and a set of math models capable of simulating the Spacelab environment.

Overall control of MPS is centralized within the MPS Executive which performs the scheduling and initiation of the various functional elements which must be executed to verify a particular planning command. The executive relies on the planning language interpreter to generate a series of events which defines the sequence of activation of each math model and the input to be supplied based on parameters extracted from the user command. Although little information is available at present on the anticipated number of users during periods of peak activity, it is not unreasonable to assume that the executive must be capable of supporting over 100 active remote terminal users concurrently.

The Terminal Management Subsystem (TMS) is envisioned as a commercially available teleprocessing package capable of interfacing with the remote terminals over telecommunication lines and performing such basic functions as message editing, queueing, and routing. The Display Formatter accepts MPS executive generated output messages from the TMS and reformats the information contained therein into a predefined display format. Location of the display formatter remotely in intelligent terminals offers the advantages of minimized line traffic and reduced load on the central MPS CPU.

In the batch mode, the Terminal Management Subsystem (TMS) is replaced by a Remote Job-Entry (RJE) capability which introduces input to the MPS from a batch input device (card reader, tape, disk). The Display Formatter is replaced by a compiler-like subsystem capable of producing descriptive error and warning messages suitable for hard copy printout.

The Data Base Manager is assumed to be a commercially available DBMS package which is compatible with the TMS selected. Lease or purchase of such packages from a vendor not only reduces software development costs, but allows programming resources to be concentrated on the primary problem of developing the MPS unique software. If the decision is made to develop the TMS and DBMS functions within NASA, it is recommended the development be completed prior to development of the other MPS modules in order to provide a firm base for such development which will not change dynamically.



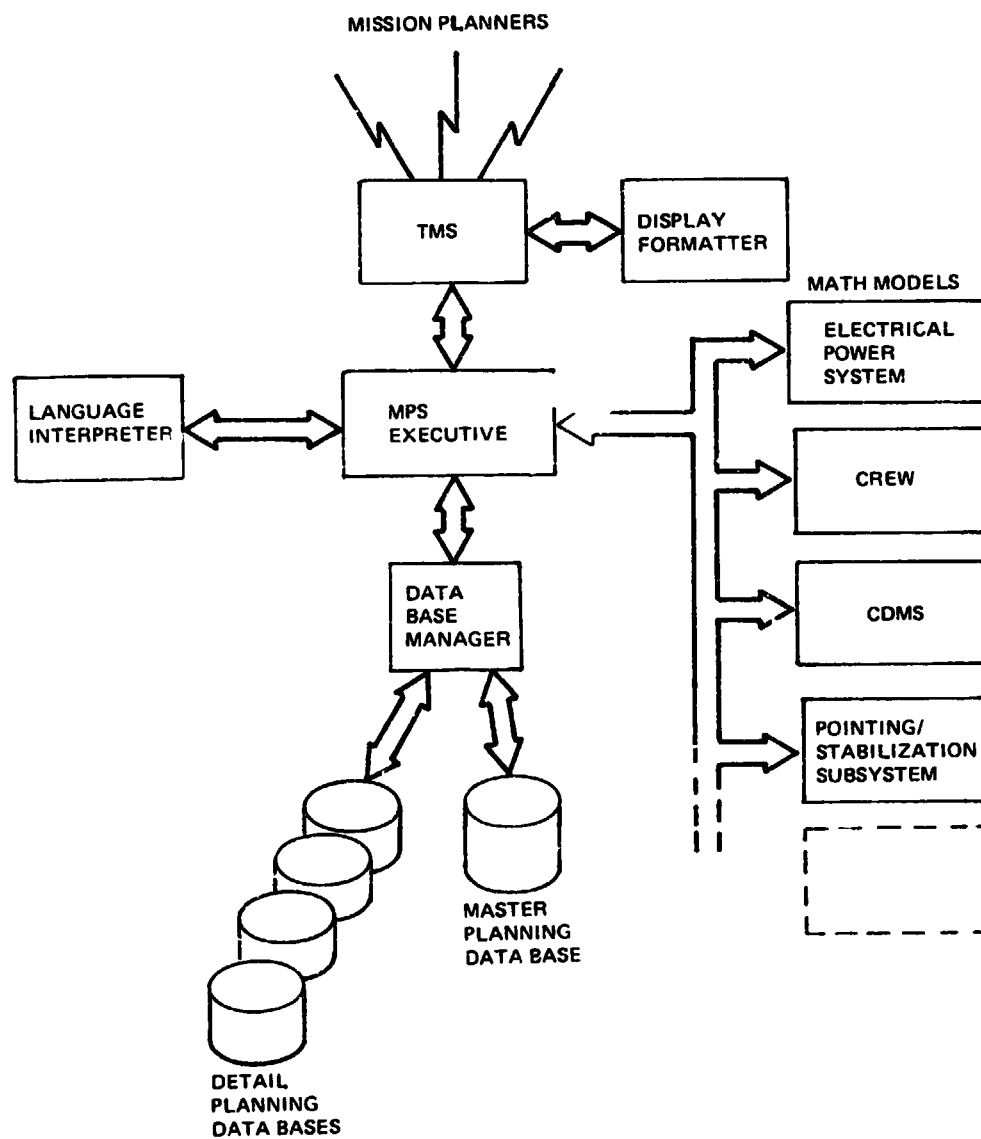


Figure 4-4. Baseline Mission Planning Systems Architecture

The Data Base Manager provides the facility for on-line storage of the massive amounts of data produced by the planning activity. Variable length strings of data must be stored, expanded and otherwise manipulated by the MPS in the process of refining a gross mission profile into an integrated detail mission plan. Each user is provided the capability of creating his own data base either by copying an existing portion of the master or by entering data direct<sup>1</sup> into the system. The flexibility of massaging these detail data bases in a responsible, interactive manner must be provided as well as the facility of merging the resulting verified product back into the master mission data base.

The mathematical models are capable of stimulating the responses generated by the various elements which make up the onboard environment. Stimuli developed by the language interpreter are used to activate each model in the proper sequence for incorporation into the mission plan. Models exist for such Spacelab functions as power distribution, CDMS, telemetry and others. Each model is capable of retaining an awareness of its state as produced by previous stimuli from earlier commands and/or interaction with interrelated models. Each model is constructed such that should an event occur that caused the state being developed to extend beyond the bounds of the constraints of the Spacelab element being modeled, an immediate notification of the error condition is made to the user. An example of such an error condition might be the performance of two or more operations concurrently whose additive effects would overload the electrical power system of the Spacelab. After notification, the planner can adjust the sequence of activities in order to include the required operation on a non-interference basis.

#### 4.4.3.2 MPS Utilization

As was discussed previously, the MPS is utilized in two distinct modes: (1) pre-mission planning in development of the detailed mission timeline, and (2) contingency operation planning during the mission.

##### Premission Planning

In development of the detailed mission plan, detailed definition of experiment operational characteristics and constraints will be included in the math models described previously. A preliminary mission timeline will be generated and placed into the data base. This preliminary timeline will provide the planning base to be utilized in establishing the detailed mission timeline. Working from the planning base, the mission planner will extract a section of time and refine it to a greater level of detail. The more detailed version of that planning increment can then be immediately verified by merging it into a copy of the master. During the merging process, the MPS assures that all interface requirements are met and, if not, notifies the planner who may then extract, alter, and repeat the merge/verify process until all constraints imposed by the math models have been satisfied.

### Contingency Operations

In the event of experiment failure during the Spacelab mission, the MPS will provide the capability for timeline update to optimize the scientific objectives of the remaining experiments of the payload. The procedure utilized for contingency operations updating of the mission plan is similar to that described for pre-mission planning and utilizes the interactive terminal capability.

#### 4.5 PI PARTICIPATION IN MISSION OPERATIONS CONCEPT

##### 4.5.1 THEME

To ensure that the scientific objectives of Spacelab missions are achieved, the PI must participate in the mission operations concept.

##### 4.5.2 CONCLUSIONS

The PI's involvement in mission operations must include the following:

- Generation of mission operations and preprocessing software requirements in support of experiment.
- Monitoring of experiment operation either onboard or in the POC during mission.
- Detailed experiment definition and constraints for mission planning purposes.
- Experiment Flight Applications software definition.

##### 4.5.3 DISCUSSION

The change in emphasis from operational considerations to scientific data gathering/evaluation, within the mission operations concept, will require that the PI participate in the mission operations and support functions. This participation will primarily be in the areas of software requirements and mission support.

##### 4.5.3.1 Software Requirements

Although the PI will not participate in mission operations software development, he must define the software requirements needed for his active participation through the POC. This definition will include the correlation between uplink and downlink data and his experiment, any unique requirements of this experiment, and display support requirements.

In addition to POC software, the PI must provide mission planning considerations to the Mission Planning System. These considerations will include such items as power-on/warmup timing, power-down timing, pointing constraints, lighting constraints, storage data capacity, and the description of the CDMS Experiment Flight Applications software related to his experiment.

The data preprocessing facility software must also be supplied with requirements by the PI. These requirements will include data bases,

format/parameter correlation, unit conversion constants, and decommutation information. The output format and media of the data must also be specified to be compatible with the PI's software and facility used for detailed scientific data analysis.

#### 4.5.3.2 Spacelab/PI Interface Concepts for Mission Support

For PI involvement in mission support, four concepts will be utilized. These are:

- Ground-based experiment control
- Onboard experiment control
- Automatic control
- Onboard experimenter control with ground monitoring

These concepts are illustrated in Figures 4-5 and 4-6.

##### Ground-Based Experiment Control Concept

This concept will be utilized primarily for pallet-only Spacelab missions. In this concept, the PI will have responsibility for control and monitoring of his experiment from a console in the POC. Communication between the PI and the experiment will be provided via the downlink and uplink capabilities. Software to support the PI's console must be provided by the POC.

##### Onboard Experiment Control

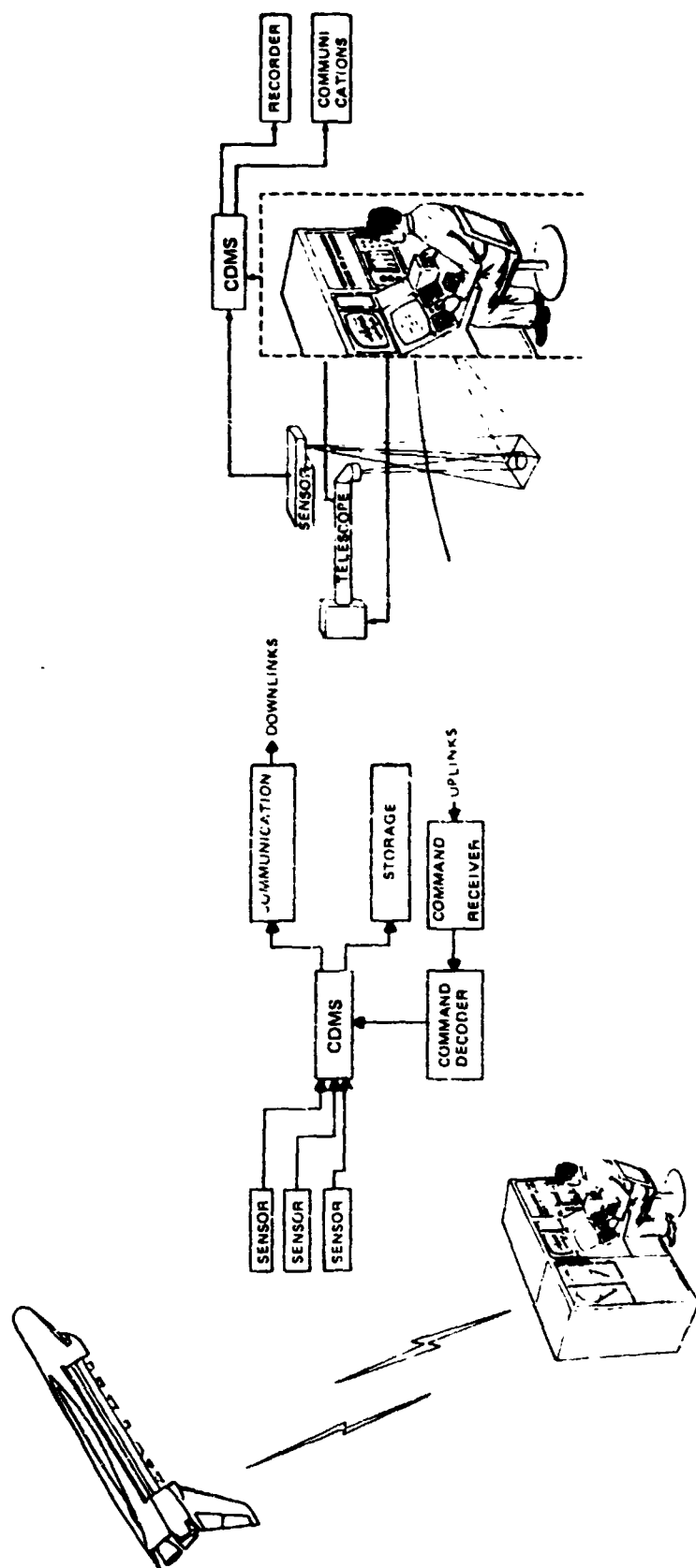
This concept will require that the PI or his representative be a member of the Spacelab crew. In this concept, the PI communicates with his experiment through the onboard consoles and the flight applications software of the CDMS.

##### Automatic Control Concept

This concept will utilize the Experiment Flight Applications software for control and monitoring of experiment operation. Utilization will be limited to simple experiments requiring a minimum of support.

##### Onboard Experimenter Control with Ground Monitoring

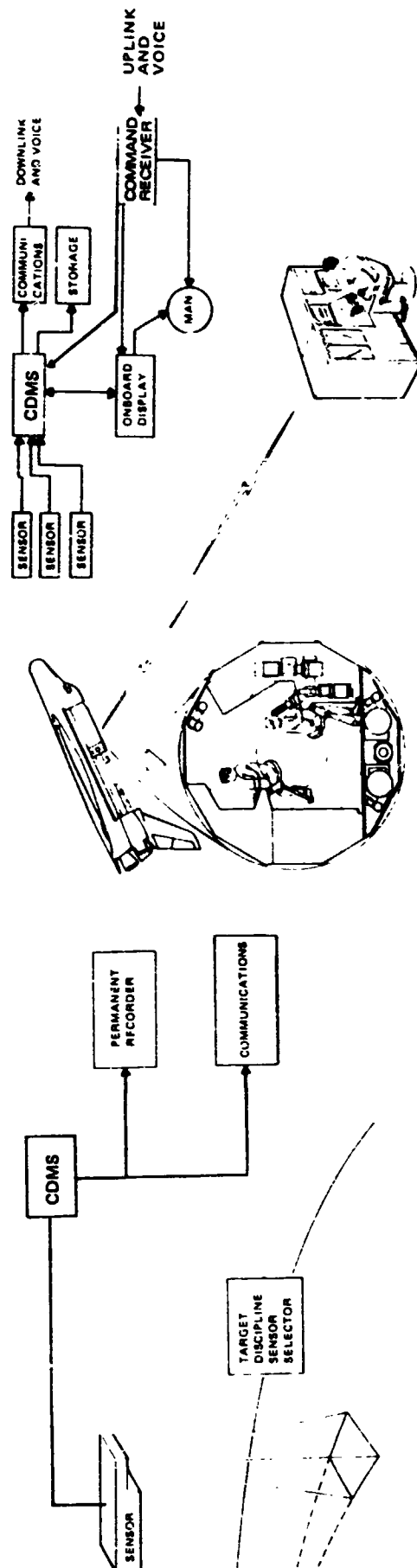
This concept will provide a team approach for complicated experimenter's operation. This approach will require voice communication between POC and onboard experimenter.



EXPERIMENT CONTROL - ONBOARD EXPERIMENTER

EXPERIMENT CONTROL - GROUND BASED EXPERIMENTER

Figure 4-5. PI Mission Support Concepts, Options 1 & 2



EXPERIMENT CONTROL -  
ONBOARD EXPERIMENTER INTERACTION  
WITH GROUND MONITORING

EXPERIMENT CONTROL -  
AUTOMATIC INTERVENTION

Figure 4-6. PI Mission Support Concepts, Options 3 & 4

#### 4.6 STIL SUPPORT REQUIREMENTS

In the course of this study, no definite requirements in support of mission operations have been established for the STIL. It is felt that mission operations software requirements will be fulfilled by the mission operations personnel on their own computers within the POC. However, should it occur that the POC computers and the STIL computers are compatible and are located in the same facility, the mission operations basic functions of compiling and assembling could be performed on the STIL equipment.



# TASK 5: SOFTWARE TEST AND INTEGRATION REQUIREMENTS

## 5

SECTION		PAGE
5	TASK 5: SOFTWARE TEST AND INTEGRATION REQUIREMENTS . . . . .	5-1
5.1	TASK 5: SUMMARY . . . . .	5-1
5.1.1	Conclusions . . . . .	5-1
5.1.2	Objectives/Study Approach . . . . .	5-1
5.2	STIL REQUIREMENTS SUMMARIZATION . . . . .	5-5
5.2.1	Theme . . . . .	5-5
5.2.2	Conclusions . . . . .	5-5
5.2.3	Discussion . . . . .	5-5
5.2.3.1	Processing Load Requirements . . . . .	5-5
5.2.3.2	Operational Modes/Development Tools Requirements . . . . .	5-6
5.3	DEFINITION OF STIL OPERATIONAL MODES/DEVELOPMENT TOOLS . . . . .	5-11
5.3.1	Theme . . . . .	5-11
5.3.2	Conclusions . . . . .	5-11
5.3.3	Discussion . . . . .	5-11
5.3.3.1	Realtime Mode Discussion . . . . .	5-13
5.3.3.2	Batch Processing Mode . . . . .	5-21
5.3.3.3	Supportive Mode . . . . .	5-30
5.3.3.4	STIL Data Base Definition . . . . .	5-37
5.3.3.5	STIL Load Requirements . . . . .	5-42
5.4	STIL MODELING ANALYSIS . . . . .	5-45
5.4.1	Theme . . . . .	5-45
5.4.2	Conclusions . . . . .	5-45
5.4.3	Discussion . . . . .	5-45
5.4.3.1	Model Description . . . . .	5-45
5.4.3.2	Model Input Definition . . . . .	5-48
5.4.3.3	Model Utilization . . . . .	5-48
5.4.3.4	Model Result Summary . . . . .	5-59
5.5	STIL CONFIGURATION ANALYSIS . . . . .	5-61
5.5.1	Theme . . . . .	5-61
5.5.2	Conclusions . . . . .	5-61
5.5.3	Discussion . . . . .	5-61
5.5.3.1	Functional Configuration Analysis . . . . .	5-62
5.5.3.2	Candidate STIL Configuration Concepts . . . . .	5-66
5.6	STIL DEVELOPMENT PLAN ANALYSIS . . . . .	5-73
5.6.1	Theme . . . . .	5-73
5.6.2	Conclusions . . . . .	5-73
5.6.3	Discussion . . . . .	5-73
5.6.3.1	Phase I Development . . . . .	5-75
5.6.3.2	Phase II Development . . . . .	5-75

## TASK 5: SOFTWARE TEST AND INTEGRATION 5 REQUIREMENTS

### 5.1 TASK 5: SUMMARY

The requirement for a dedicated facility (STIL) to support the development, integration, and delivery of CDMS software was established during Definition of Spacelab Experiment Software Development Concepts as described in Section 2. It was further established in Test and Checkout Software concepts (see Section 3) analysis that the STIL must provide capabilities to be utilized in maintenance and support of CDMS and EGSE Test and Checkout software. To satisfy the requirements levied by these previous study activities, a preliminary STIL definition was performed. Although the definition is preliminary, it will provide the basis for more detailed definition in future study activity.

#### 5.1.1 CONCLUSIONS

As a result of analyses conducted during the software test and integration task (Task 5), the following conclusions have been established:

- A host computer complex with CPU capability of approximately three MIPs and memory capacity of three megabytes is required to handle the processing load established in this study and to provide adequate growth capability.
- Selection of the final STIL hardware configuration requires additional detailed analysis.
- A sophisticated terminal controlled data base management system will be required to support software management configuration and software development concepts.
- The STIL must provide for both realtime and batch processing services to support development requirements.

#### 5.1.2 OBJECTIVES/STUDY APPROACH

The objectives of the software test and integration task were to: (1) establish a baseline understanding of the STIL requirements in the areas of operational environment and development tools, (2) develop a model of the STIL for determining the domain of CPU and memory requirements of the host computer, (3) develop representative STIL configurations, and (4) establish a preliminary STIL Development Plan.

In achieving these objectives, IBM established a systematic study approach. This approach, with the interrelationships among the elements, is shown in Figure 5-1.

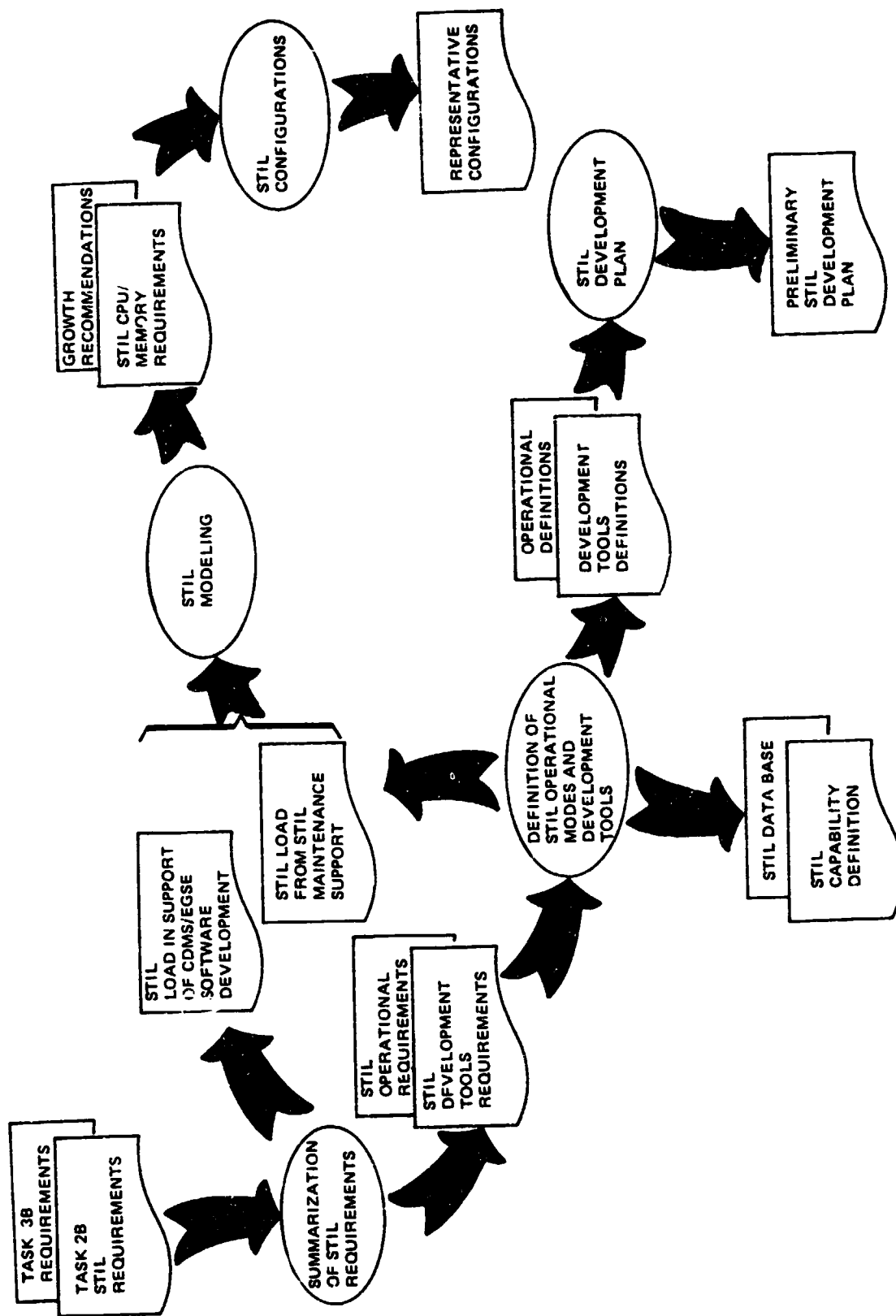


Figure 5-1. Software Test and Integration Task Flow

The systematic approach utilized in satisfying the overall task objectives required that the study of five major areas be conducted. These areas are:

- STIL Requirements Summarization
- STIL Operational Modes and Development Tool Definition
- STIL Modeling Analysis
- STIL Configuration Analysis
- STIL Development Plan Analysis

The combined outputs of these studies resulted in satisfaction of study objectives. Each of the areas are discussed in detail in subsequent paragraphs.

In defining the requirements of the STIL, the following groundrules were established to aid in bounding the study effort:

- A CDMS simulator will be resident in the STIL (includes all CDMS hardware).
- The subsystem computer, the experiment computer, and the backup computer and associated input/output of the CDMS will be common.
- There will be no communication among CDMS computers.

## 5.2 STIL REQUIREMENTS SUMMARIZATION

### 5.2.1 THEME

The requirements the STIL must support for Experiment Flight Application software, subsystem software, and the EGSE software were established in the Experiment Software Development Concepts (Task 2B) and the Spacelab Test and Checkout Software Concepts (Task 3B) studies. Prior to performing the STIL definition, these requirements were compiled as input drivers to the remaining tasks of the STIL analysis.

### 5.2.2 CONCLUSIONS

The summarization of requirements resulted in the following conclusions:

- A total of 350 runs/day must be supported by the STIL. Runs will vary from maintenance type to full Spacelab on-orbit simulations.
- Experiment flight applications software development activities will generate 70% of daily load on STIL.
- STIL must support realtime and batch processing environments.
- STIL must provide state-of-the-art software development tools and supportive software to meet the challenges of the Spacelab software development environment.

### 5.2.3 DISCUSSION

In defining the STIL, the requirements have been divided into two classes--each of which is critical in establishing the baseline STIL definition. These classes are processing load (runs/day) requirements to support development activity, and operational environment/development tool requirements.

#### 5.2.3.1 Processing Load Requirements

The analysis performed during Task 2B indicated that the maximum software development burden on the STIL would occur in 1981. During that year, a projected total of 416,520 CDMS experiment applications instructions must be developed and will require a total of 250 runs/day on the STIL. In addition, the processing requirements associated with maintenance and support of the test and checkout software were determined, during Task 3B, to require 62 runs/day on the STIL.

The maximum processing load the STIL must support, as determined in the study, is summarized on a runs/day basis in Table 5.1. The load has been distributed among the functional capabilities which must be provided by the STIL. The activities included within the functional capabilities are described below:

Software Management - activities necessary to maintain source code, configuration management, statistics, and software release data base.

Software Implementation - the use of development tools (e.g., compilers/assemblers/link edits, data reduction, interpretive simulation, functional simulation, and CDMS simulation) for designing, developing, debugging, and testing of STIL-supported software.

Software Verification - the activities necessary to assure product integrity through the utilization of interpretive simulation, CDMS simulation, and data reduction tools.

Software Integration - the tasks necessary to build flight software packages and sets and perform overall testing through use of such tools as CDMS simulation and data reduction.

Software Maintenance - the tasks necessary to maintain and modify Operating Systems, Applications, and Test and Checkout software.

System Maintenance and Utilities - the activities which are considered germane to computer center operations, such as save/restore activity, software diagnostic of hardware, sorts, merges, tape copy, tape dump, and STIL scheduling.

Model Maintenance - the tasks required to maintain and update the environment models required for simulation capabilities.

As can be seen in Table 5.1, a daily load of 350 runs/day must be supported by the STIL. Of the total, 70% of the runs are made in support of experiment flight applications software development, thus making flight applications software the principal STIL user.

#### 5.2.3.2 Operational Modes/Development Tools Requirements

To support the development requirements established during Tasks 2B and 3B of this study, the STIL must provide the operational capabilities to allow effective utilization of STIL resources. The operational modes established in order to provide all required capabilities and the development tools utilized within each mode are summarized in Figure 5-2. As can be seen, the operational modes are:

Supportive Mode - provides development tools and capabilities required to support CDMS and EGSE software management and to allow efficient utilization of STIL host computer resources.

Realtime Mode - provides software development in an interactive realistic environment. This is accomplished by executing the code under development on the CDMS simulator and simulating the total environment on the STIL host computer.

Table 5.1. Processing Load Requirements Summary

FUNCTION	RUNS PER DAY		
	FLT. APPLICATION	TEST & CHECKOUT	HOST
<b>SOFTWARE MANAGEMENT</b>			
- Configuration Mgmt/Statistics	10	10	
- Automated Release	1	.1	
<b>SOFTWARE IMPLEMENTATION</b>			
- CDMS Assemble/Compile/Link	58		
- Interpretive Simulation	6		
- Functional Simulation	26		
- CDMS Simulation	13		
- Design Analysis Simulation	9		
- Data Reduction	39		
<b>SOFTWARE VERIFICATION</b>			
- Interpretive Simulation	30		
- CDMS Simulation	20		
- Data Reduction	38		
<b>SOFTWARE INTEGRATION</b>			
- System Build (Link)	.1	1	
- System Test	.1	1	
<b>SOFTWARE MAINTENANCE</b>			
- Compile/Assemble/Link		18	
- Functional Simulation		9	
- Interpretive Simulation		9	
- Data Reduction		14	
<b>SYSTEM MAINTENANCE &amp; UTILITIES</b>			8
<b>MODEL MAINTENANCE</b>			22
- Compile/Link			8
<b>SUBTOTALS</b>	250	62	38
<b>TOTAL RUNS/DAY</b>	350		

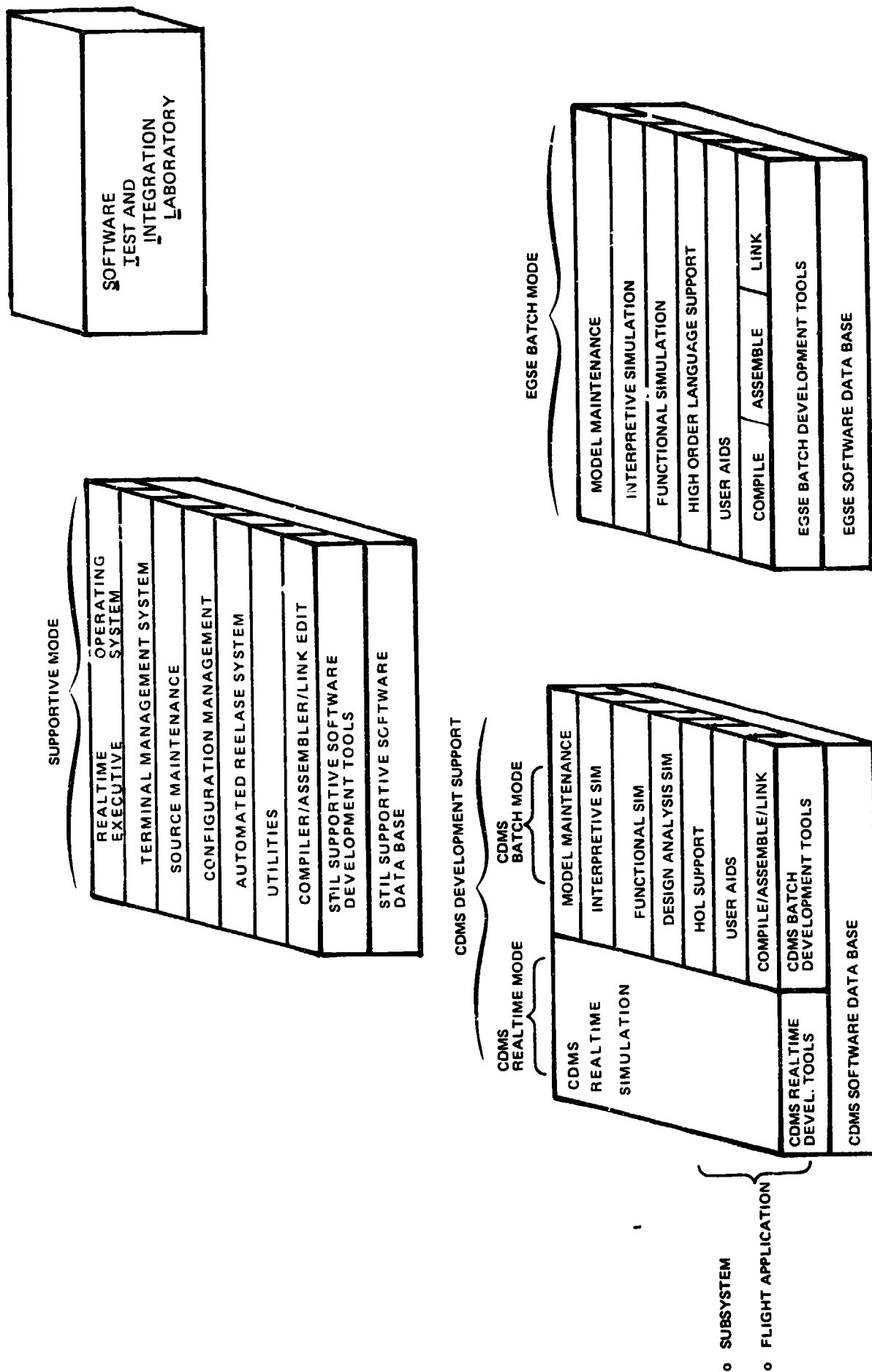


Figure 5-2. STIL Operational Modes and Development Tools



Batch Processing Mode - provides normal computer center data processing environment.

Of the above operational support modes, only the realtime mode is different from a normal computer center operation. The realtime simulation requirements were established during Task 2B and require the capability to execute flight applications software within an actual CDMS configuration. This simulation must provide a high fidelity environment for verification of the experiment flight applications software prior to delivery.

Realtime simulation provides the medium of development and testing which allows the development schedules to be achieved. As was determined during Task 2B, the maximum number of flight sets to be delivered occurs during 1985 when a total of 31 flights must be supported. To meet such delivery schedules, realtime simulation must be provided. Use of interpretive computer simulation, with a clocktime to realtime of 80 to 1, would require testing time which could not be provided within the schedules.

### 5.3 DEFINITION OF STIL OPERATIONAL MODES/DEVELOPMENT TOOLS

#### 5.3.1 THEME

In support of the design, development, integration, testing, maintenance, and delivery of software for the experiment computer, subsystem computer, and the EGSE computer, the STIL must function in three distinct operational modes and provide the development tools needed in utilization of these modes.

#### 5.3.2 CONCLUSIONS

The following conclusions were established in the course of this study:

- The STIL data base needed to support CDMS and EGSE software development will require approximately 1.3 billion bytes of storage.
- The maintenance of STIL capabilities will require eight runs/day on the host computer.
- The realtime simulation mode will require unique support functions (realtime executive, user aids, online realtime interaction, etc.).
- An extremely sophisticated simulator will be required to support the realtime simulation capability with both CDMS computers functioning simultaneously.
- To support realtime simulation, an interface device will be required between the host computer and the CDMS simulator.

#### 5.3.3 DISCUSSION

As has been previously discussed, the STIL must support: (1) a realtime mode, (2) batch processing, and (3) a supportive mode. Within each mode development tools must be provided to support the STIL user in development of CDMS and EGSE software. An additional STIL supportive function, the STIL data base, must provide the data required in controlling and reporting the status of software development activities.

Within this study task, the operational modes and associated development tools will be defined. In addition, the STIL data base and processing load on the STIL to maintain STIL capabilities will be established. Figure 5-3 depicts the operational modes and the related development tools. The following paragraphs will address each mode individually and define the development tools as indicated in the figure.

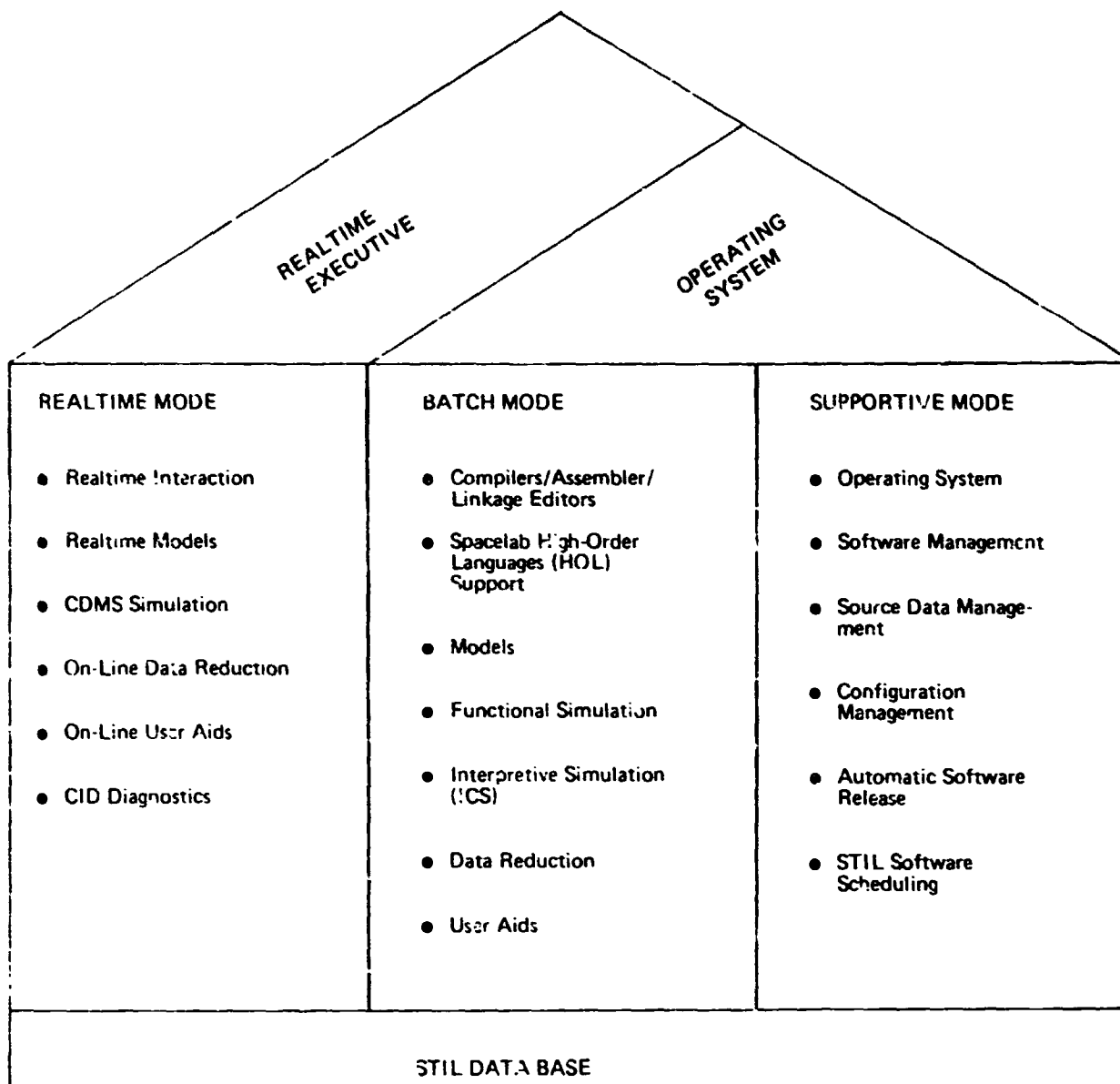


Figure 5-3. Operational Modes and Software Development Tools

#### 5.3.3.1 Realtime Mode Discussion

The realtime mode of the STIL will provide the capability to test the Spacelab subsystem and experiment computers' software in a realtime environment. Within this mode, the CDMS will be utilized and will be interfaced with the STIL host computer through a special host/CDMS Interface Device (CID). The host computer will provide the simulation of the environment needed to support the operation of the flight software. This environment will include high fidelity models of Spacelab subsystems and experiments to ensure proper interface between flight software and Spacelab hardware prior to the actual hardware/software integration. The functional diagram of the realtime testing configuration is shown in Figure 5-4.

The support elements provided during the realtime testing mode are shown in Figure 5-3 and are discussed in the following paragraphs.

##### Realtime Executive

Execution of realtime simulation necessitates that the operating system of the STIL host computer have features not found in a normal computer center operating system. As indicated in Figure 5-3, the realtime executive is envisioned as an addendum to the host operating system while retaining the capabilities to support both batch and supportive modes when excess CPU and memory is available during realtime testing.

The analysis conducted during this study has defined the following preliminary requirements which must be satisfied by the realtime executive:

- Maintain task management control over host computer software execution within specified time frame.
- Monitor execution of Spacelab CDMS software.
- Support realtime user aids and realtime interaction requirements.
- Provide interfaces between host software and CDMS interface device for I/O control.
- Perform checkpoint and data gathering functions.
- Maintain statistics pertaining to host computer facilities utilization during realtime testing.
- Provide realtime access methods for online storage utilization.
- Provide compatibility with normal host computer operation system to provide batch/supportive roles of STIL while executing realtime simulations.
- Support "hands-on" control language for user interface.

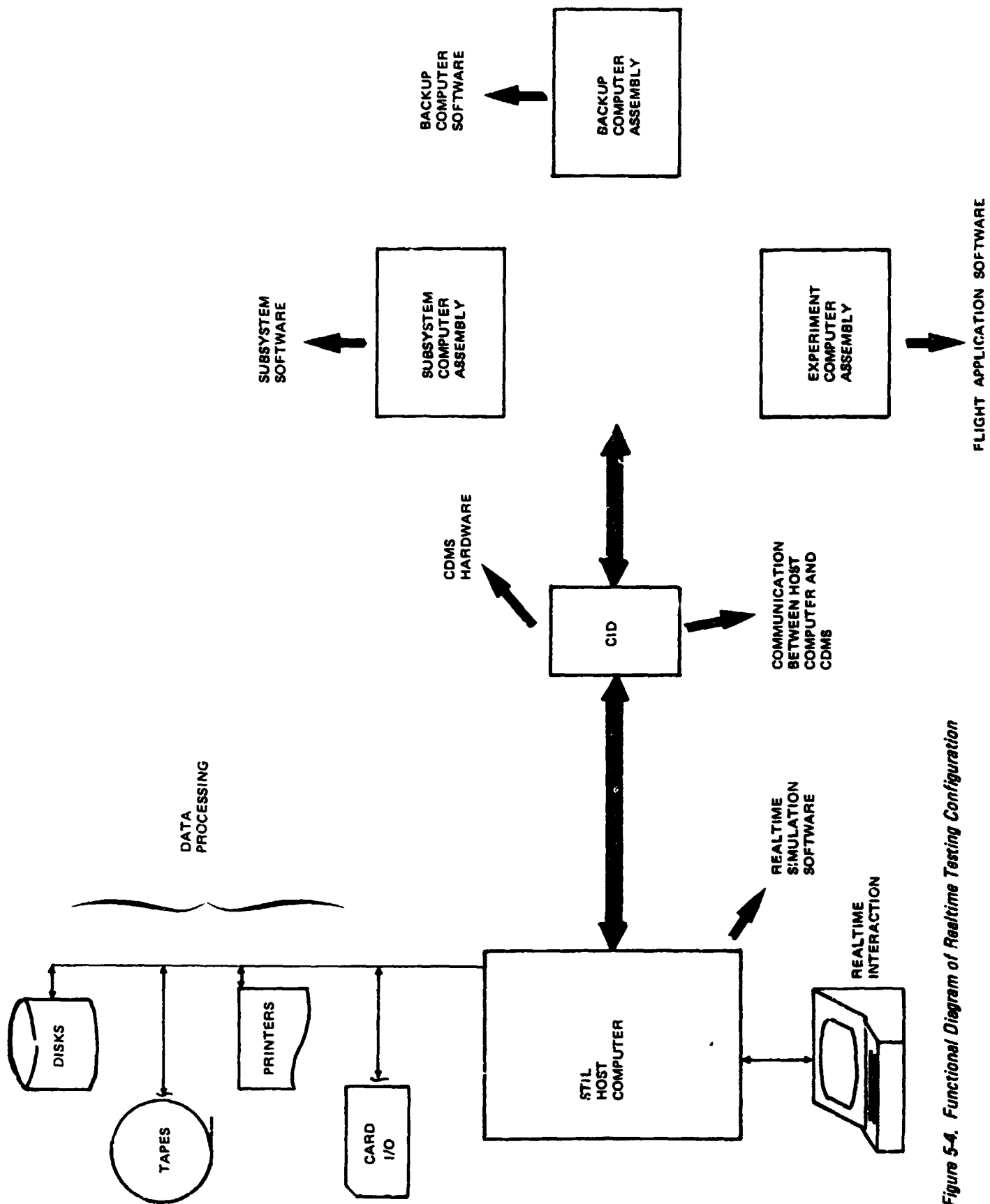


Figure 5-4. Functional Diagram of Realtime Testing Configuration

### Realtime Interaction

During the execution of a realtime simulation within the STIL, real-time interaction between the STIL user and the simulation must be provided. This interaction will be provided through use of a realtime terminal support software package within the STIL. The principal advantage gained through realtime interaction is that the user can assess the performance of the simulation, make realtime decisions regarding the status of the test, and take corrective action in the event of problems. In short, the user has the advantages of a "hands-on" development environment.

The realtime interactive software will provide the following capabilities to the user:

- Initialize simulation
- Select realtime user aids to be exercised during run.
- View selected parameters during run.
- Terminate/restart runs.
- Obtain graphical representations, during run, for online realtime analysis.
- Insert error conditions into models during realtime.
- Control data recording/analysis.

An additional interactive capability must simulate the PI involvement from the ground during the mission. The realtime interaction software must provide display and command capabilities which would exist within the Payload Operations Center (POC) to the PI. The STIL user thus will be allowed to simulate the control and monitoring of experiments by the POC.

As can be seen from the above discussion, the realtime interaction capability is an essential element in overall system test and verification in that the STIL user is provided detailed visibility into simulation activity and thus can fully evaluate the operation of the subsystem and/or CDMS flight applications software in a realistic test environment.

### Realtime Models

Digital models of the Spacelab environment will be critical to the STIL's ability to support the realtime testing mode. Since the realtime testing mode provides the primary verification test bed for CDMS software (Subsystem and Experiment), it is mandatory that the models used in the simulation be high fidelity representations of the elements of the Spacelab. The high fidelity requirement will necessitate that, as the hardware configurations change, the models utilized in realtime testing must also be updated. This continuing development/maintenance activity will create a load factor on the STIL resources throughout the Spacelab program.

The realtime models will provide the building blocks needed in providing a realtime simulation capability. The models will be combined through the use of STIL software to create the host simulation software system. The models to be provided by the STIL for realtime testing must include:

- Instrument pointing system model
- Models for each experiment package
- EGSE model
- Shuttle Orbiter Interface Model (uplink/downlink/PSS)
- Model of functions of subsystem computer software
- Model of functions of experiment computer software

The above list is preliminary and can be updated as more specific information is obtained regarding Spacelab subsystem and experiment configuration and interface requirements.

The realtime models, being of a mathematical form, will be developed and maintained utilizing a high-order language. A library of these models will be maintained within the STIL data base for rapid accessibility in configuring simulators needed for each Spacelab payload.

#### CDMS Simulation

The CDMS simulation capability will be provided through the host simulator software, the CDMS Interface Device (CID), and the actual CDMS computer assemblies containing the flight software undergoing test.

Because the complete CDMS hardware will reside within the STIL, it has been assumed that the capability must exist to support realtime testing of both the experiment flight applications software and the subsystem computer software simultaneously. In addition, it has been assumed that the STIL must support individual computer software testing while simulating the functions of the other computer. The capability to support two simultaneous simulations will require a highly sophisticated host simulator and an equally sophisticated CID. Although all models have been assumed to be digital during this analysis, the existence of the RAUs within the CDMS will provide the capability to add analog models of experiments and subsystem components to the STIL if required.

Utilization of the onboard CRT/consoles during realtime testing will require that the console outputs be routed to the CDMS Interface Device and host computer software. This requirement will allow the simulator to monitor activity between the CRT/console and the CDMS software and to reconfigure

the simulator to reflect changes in the CDMS computer software. The CRT/console capability will allow the software developer to modify memory locations, modify software flow, inhibit experiment interfaces, and other activity directly affecting simulator operations. Failure to monitor CRT/console CDMS software activity could cause failure of runs, generation of bad data, and could result in significant rerun burden on the STIL. To monitor the interface activity, the simulator must:

- Include an interpreter of the onboard control and display language.
- Recognize function key input requiring modification of model execution.
- Correlate CDMS software modifications with simulator actions.
- Perform necessary reconfiguration of the simulator.

The capability to support individual testing of either the Flight Applications software or the subsystem software will require that the host simulator provide a model of the system elements not operational during the test. This model will be a functional representation and will be utilized primarily to provide the capability to perform testing of individual systems in the event of hardware/software problems within the STIL. The individual system testing capability is shown in Figure 5-5.

Since the flight CDMS contains three computers (subsystem, experiment, and backup) and a mass memory for overlaying contents of computer memory, the STIL realtime simulation mode must provide the capability to test the utilization of the backup computer in both the subsystem and experiment role. In addition, capability to test the onboard mass memory overlay capability in a realtime environment must be provided.

#### Online Data Reduction

In support of the realtime interaction capability, online data reduction must be provided. The data thus provided will allow the STIL user to monitor status of the test and make necessary decisions regarding validity of testing. The data presented to the user must be restricted in order to maintain a realtime environment; therefore, the capability must exist to select the desired parameters before initiating the test or to select new parameters during the test.

The online data reduction software must be designed to allow rapid re-definition of data stream elements. As Spacelab mission payloads vary, the telemetered data will differ and, therefore, the online data reduction software must be reconfigurable, through realtime interaction capability prior to the test initiation, to ensure compatibility between data reduction software and data stream.



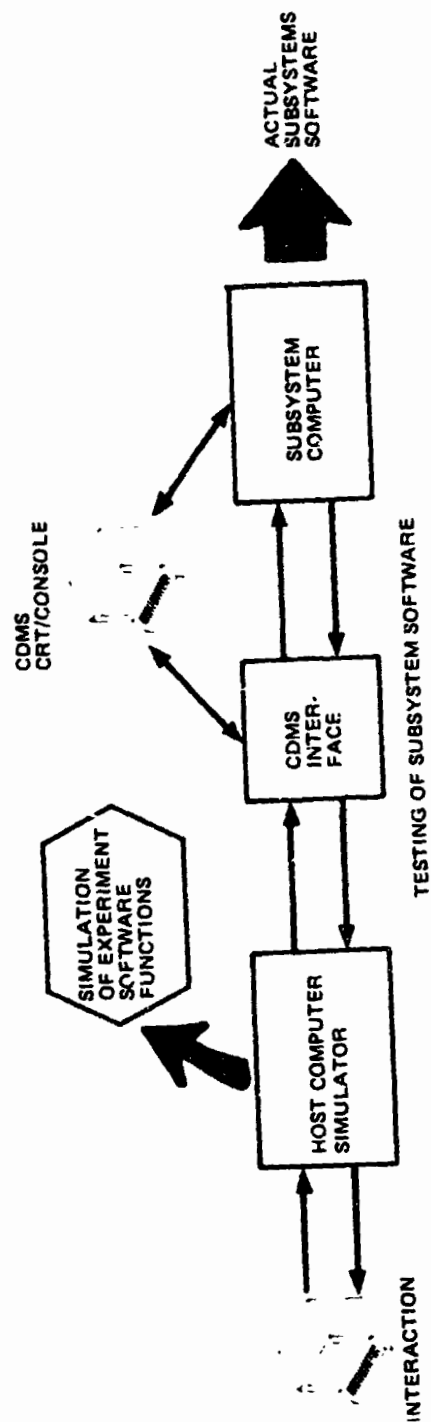
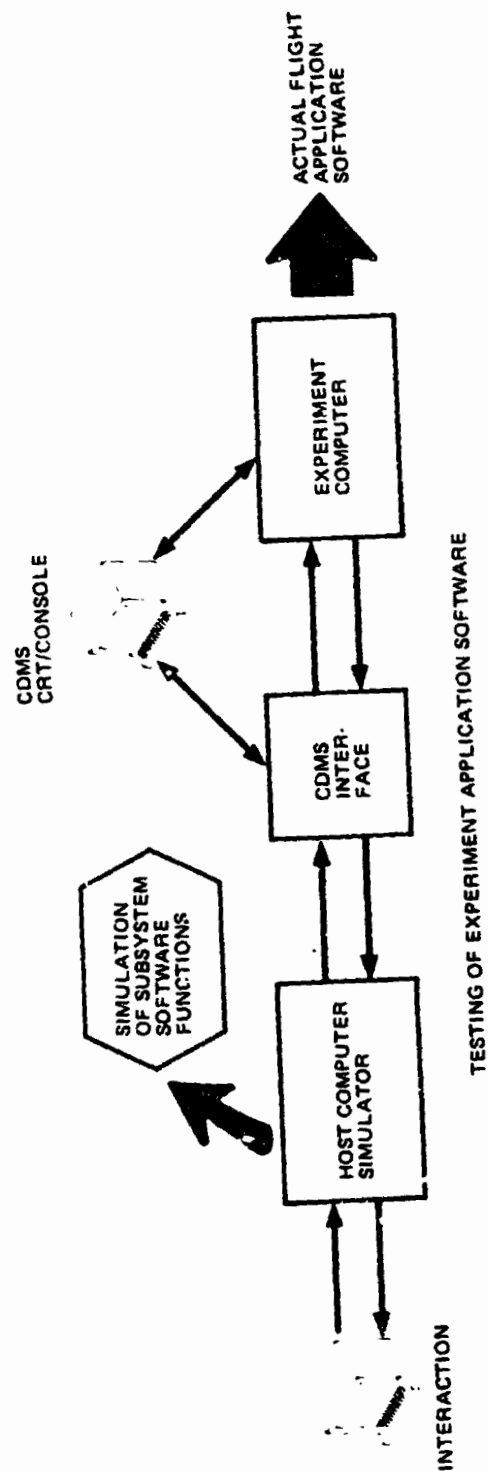


Figure 5-5. STIL Realtime Individual Computer Simulation

The output of the online data reduction will be either in engineering units or raw data form and can be presented on either the display or the printer. The user must have complete flexibility in output format and content to provide maximum productivity.

#### Online User Aids

To perform testing of flight applications and subsystem software on the STIL, the user must be provided with user aids to provide visibility into CDMS software operations. The user aids provided within the realtime testing mode require unique host computer software to interface with the CDMS Interface Device for retrieval of test data.

The user aids identified for realtime testing support can be invoked either through the display or through user setup cards prior to run. The ability to modify or cancel selected user aids during realtime runs must be provided.

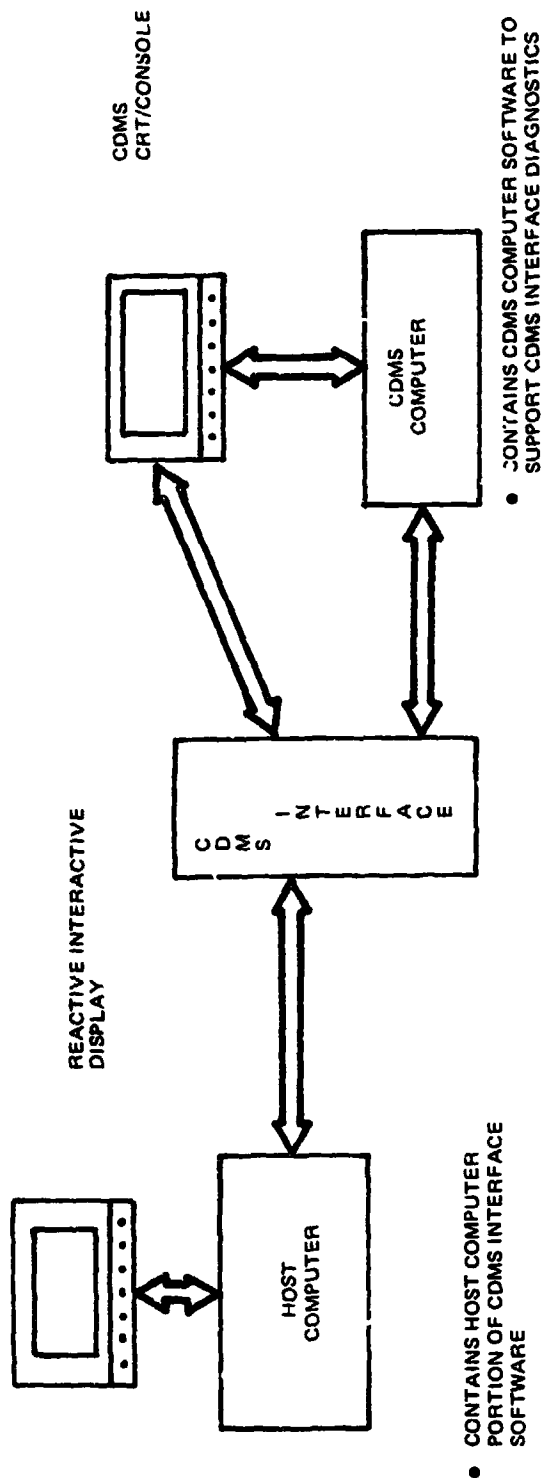
User aids identified which must be provided by the STIL realtime software system are:

- Memory load/verify/dump of CDMS computer under test
- Snaps on contents of specific CDMS computer locations
- Instruction trace
- Single step of CDMS computer execution
- Checkpoint/restart
- Modification/patch of CDMS memory contents
- Start/stop of run on input parameter

These user aids will require that the CDMS Interface Device support the access to CDMS computer control lines by the host software.

#### CID Diagnostics

In support of the realtime operational mode, the STIL user must have the capability to ensure correct operation of the CID and the CDMS. To provide this capability, diagnostic software will be provided to perform CID testing. This software will perform both functional and detailed levels of testing under control of the STIL user. A functional description of the use of the CID diagnostics is shown in Figure 5-6.



- CID AND CDM'S AUTOMATED POWER ON/OFF
- TWO MODES OF TESTING (FUNCTIONAL/DETAILED)
- TEST MODE UNDER CONTROL OF REALTIME INTERACTION DISPLAYS
- FUNCTIONAL TEST MODE UTILIZED BEFORE EXECUTION OF REALTIME TEST
- DETAILED TEST MODE UTILIZED TO PERFORM TROUBLESHOOTING
- TEST CONSISTS OF COMMAND/RESPONSE BETWEEN HOST SOFTWARE AND CDM'S SOFTWARE TO EXERCISE ALL CDM'S INTERFACE FUNCTIONS

Figure 5-6. CID Control & Test Utilization

#### 5.3.3.2 Batch Processing Mode

In addition to the realtime mode, the STIL must support a normal computer center processing environment. Within this environment, the STIL user can submit computer jobs either remotely through terminals or over-the-counter. Each job submitted is entered into the job stream manager of the host computer operating system and executed as resources needed to perform the job become available. Upon completion of the processing job, output will be routed to the user.

Whereas the realtime simulation provides the primary software debug and verification environment, the batch processing mode supports the primary software design and development tools needed in maintaining and developing software for EGSE, experiment flight applications, and subsystem software.

The capabilities and development tools provided by the batch processing mode will be discussed in the following paragraphs.

##### Compilers/Assemblers/Linkage Editors

To support the STIL user performing modeling, data reduction/analysis, and simulation on the host computer, compilers/assemblers/linkage editors for the host computer must be provided. These support software packages must be provided with the host computer and will include such packages as Fortran, PL/I, macro assemblers, and linkage editors. No development activity will be required in utilization of these packages.

##### Spacelab High-Order Language (HOL) Support

The CDMS software and EGSE software will be developed utilizing high-order languages. Since these languages have not been selected, an assumption has been made that unique high-order languages, not normally supported by a host computer, will be required. It is expected that ESRO will make available the compilers to support these languages, and they will be compatible with the STIL host computer.

Another class of high-order languages to be supported are procedure-type languages used in development of models for use in simulation. The compilers/interpreters to support these languages must be developed and maintained within the STIL host computer software system.

In support of the functional simulation capability, the HOL compilers/interpreters must provide the capability to insert programmer testing aids within the source code at the statement level to allow the programmer to utilize simulation debug aids. In addition, the compilers must provide capability to translate application software statements into host computer machine language for execution.

In addition to compilers to support the CDMS and EGSE computers, an assembler will be required to support writing software, which is either time-critical or core-limited. It is anticipated that the STIL will utilize the ESRO-developed CDMS/EGSE assemblers.

A linkage editor, included with both the compiler and assembler, will also be provided by the STIL. This linkage editor will provide the capability to combine CDMS and EGSE application modules into software packages and sets for testing and delivery.

Although the STIL will attempt to utilize the ESRO-developed HOL support tools, incompatibilities may exist between European host computers and the STIL host computer which would make this utilization difficult. Even if compatibility does exist, the STIL utilization requirements may require modifications to ESRO-developed software. An area of potential difference exists in software management techniques. It is anticipated that the support packages will have capabilities to assist in maintaining configuration management over CDMS and EGSE software within the STIL. These capabilities may not be required within ESRO and so may not be included within the support software.

#### Batch Models

Models of the Spacelab systems and operational environment will be required in order to support the simulation capability provided within the batch processing mode. These models will vary in level of fidelity from those used to support realtime testing to those of a very functional nature. The Spacelab environment model will include those elements previously discussed for realtime simulation and may very well be the same software modules.

An additional model maintained and utilized in the batch processing mode (known as the Design Analysis Simulator) must provide a mathematical representation of the subsystem and experiment computer software and will be utilized to predict software operational characteristics under varying experiment loads. The model will be driven by functional level models of subsystems or experiment elements. The results of this modeling will identify potential design and/or operational constraints early in the development cycle to allow software and/or experiment modification to be accomplished without schedule impact.

To assist in support of the stand-alone testing of application software, test models must be developed. These models will supply a fixed data input stream at a specific rate and format for driving the logic of the software undergoing test.

#### Functional Simulation

The functional simulation capability of the batch processing mode will provide the user of the Spacelab HOL with the ability to execute the CDMS software statements on the host computer in the language of the host computer. This capability will require that the HOL compiler utilized in development of CDMS software provide the option of generating object pro-

grams in either CDMS form or host computer form. In addition, to provide debug aids, the compiler must insert linkages within the source code to allow communication with the functional simulation control and diagnostic software included within the functional simulation package.

Because the functional simulation is performed in the language of the host computer, it is anticipated that the simulation can be executed at a rate greater than the actual Spacelab flight time. This execution rate has significant advantages in that maximum utilization of the functional simulation can reduce the burden on the realtime simulation capability for debug purposes. This reduction of burden on the realtime simulation capability will allow the CDMS to be dedicated more fully to verification of software.

The functional simulation capability will allow the STIL user to perform testing on timing and logic flow within his software. Because the simulation is executed within the batch processing mode, multiple users can execute simultaneously.

As can be seen in Figure 5-7, the functional simulator will consist of:

- Host computer version of CDMS software statements
- Spacelab functional environment models
- Functional simulation control/diagnostics software

The functional simulation control/diagnostic software must provide the following capabilities:

- Simulator initialization based on user input
- Execution start and job termination
- Simulator timing
- Error traps (arithmetic errors)
- Intercept abnormal end of job conditions
- CDMS operating system traces (actions taken, scheduling of tasks, etc.)
- Input/output control within simulation
- Output control for data recording

The Spacelab functional environment models will be functional representations of the hardware. This functional nature is required in order to achieve execution rates exceeding realtime. Because the functional simulation will be utilized for debug purposes, the functional level of modeling will be acceptable.

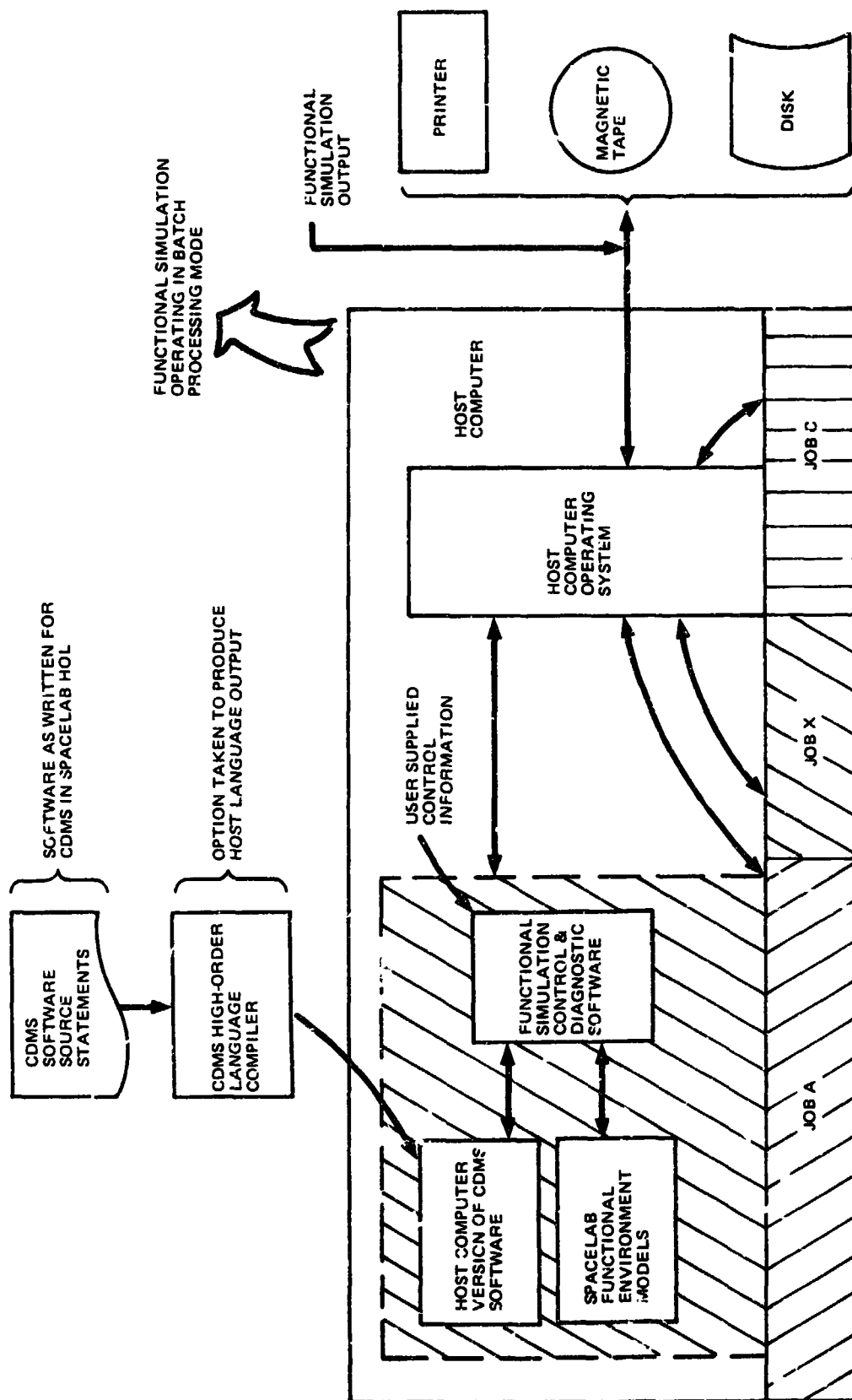


Figure 5-7. Functional Simulation Description

### Interpretive Computer Simulation (ICS)

The ICS utilized in the batch processing mode must provide the capability to execute actual CDMS software and EGSE software within the host computer. To accomplish this capability, an interpreter must be provided to decode each CDMS computer instruction and execute the host computer instructions needed to duplicate exactly the results which would be obtained in the CDMS/EGSE computer. The ICS must also provide the capability to repeat all test runs exactly in order to perform detailed timing and interaction testing among the CDMS software modules.

In addition to the simulation of CDMS computer software execution, the ICS will contain a detailed model of computer architecture and input/output which will include registers, interrupt capability, timer, and clocks. This model allows the ICS user access to contents of registers for detailed analysis of software execution and will also allow the user to insert failures during testing.

In support of Spacelab software development activity on the STIL, two distinct ICS versions are required--CDMS computer, and EGSE computer. The primary utilization of the ICS for CDMS will be in detailed verification test cases in which timing and interaction are the primary considerations. Since bit-for-bit repeatability can be obtained from run to run, complex problems can be recreated and repeated for detailed analysis. Because minimum change is anticipated, the ICS for the EGSE software will provide the only testing capability supported on the STIL for EGSE software.

The ICS must include extensive user aids for controlling the simulation and obtaining output data. These user aids must include the following:

- Logic trace
- Block or region trace
- Memory dump
- Time trace
- Memory correction or patches to CDMS/EGSE software
- Register snaps
- Stop/restart
- Cancellation of selected aids

A functional description of the ICS is shown in Figure 5-8. As can be seen, the ICS will consist of the following elements:



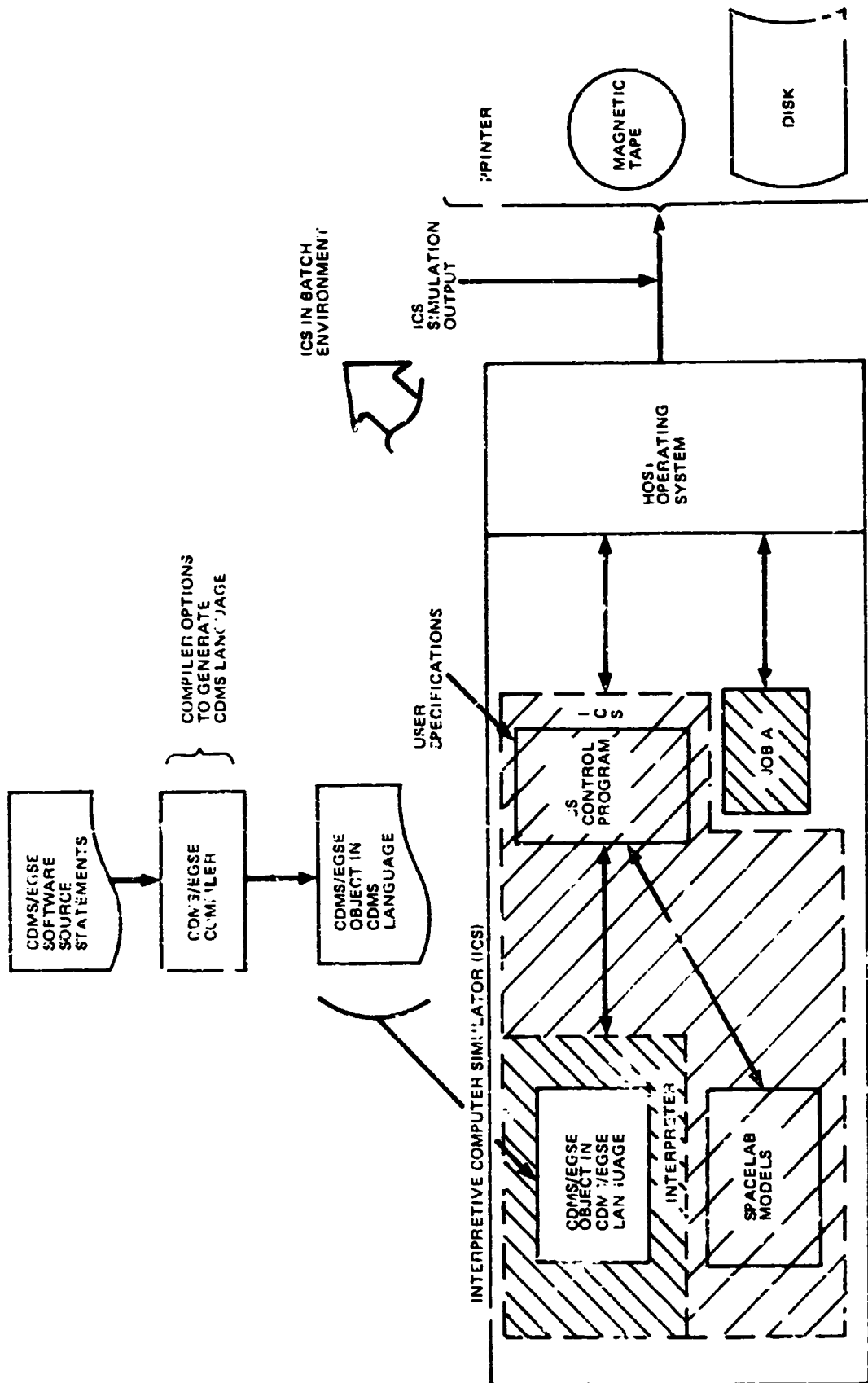


Figure 5-8. Functional Description of ICS in Batch Environment

- ICS control program
- Interpreter
- CDMS/EGSE object program in CDMS/EGSE language
- Spacelab models

The ICS control program must provide the following capabilities critical to the correct operation of the ICS:

- Simulator initialization
- Diagnostic input/output processing
- Input/output processing (OS interface)
- Timing control
- Restart processing/termination
- Control over simulator elements

The interpreter must provide the capability to access the CDMS/EGSE object code, decode and perform the operation, and ensure a bit-for-bit representation of CDMS/EGSE computer result. The interpreter must also include the control of CDMS/EGSE internal hardware logic to ensure proper flow within the CDMS/EGSE software. It is worthy to note that, if the EGSE and CDMS computers are of different types, different interpreters will be required.

The software models of the Spacelab and its environment must be of high fidelity to support verification utilization.

#### Data Reduction

Successful support of experiment computer software, subsystem computer software, and EGSE computer software development activities will require a maximum of sophistication and flexibility of data output. The data reduction and analysis software must provide the STIL user with the tools needed to retrieve, manipulate, reduce, format, and output the data obtained from the simulations.

Because the experiment flight applications software will continue to change throughout the Spacelab program, the design of the data reduction software must allow for re-definition of input stream by users. This will avoid constant modification of data reduction software from mission to mission.

The data stream being processed by the data reduction software will include that data gathered during real-time simulation. Because of the similarity of functions performed, the batch mode data reduction software must provide the baseline for development of on-line data reduction software.

As can be seen in Figure 5-9, the data reduction process can be subdivided into three major elements--input, process, and output. These elements are briefly discussed in the following paragraphs.

The input phase must contain sufficient data to identify to the data reduction software the data to be processed, how the data is to be manipulated, and the output format of the data. These input requests can be provided to the STIL via card or terminal input and must be easily understood by the STIL user to provide the necessary flexibility. Typically, this input must contain:

- Input data definition
- Selected parameters to be processed
- Format of input data
- Conversion definition
- Output media and format

The input data processed by the data reduction software will be that data collected during simulation runs on the STIL. Because of the different environments under which simulations occur, the input to the data reduction software may vary considerably in both format and volume. The data reduction software must provide for this difference through the user request facilities.

The process function of the data reduction software will consist of two major portions--control and data handling. The principal functions of the control portion will be to process input requests, to control the processing performed on the input data, and to route the output to the requested output media. The data handling portion must perform the required manipulations, formatting, and conversions necessary to satisfy the user request.

The output function will consist of the output media to which the resulting data will be routed. Within the STIL, this media will be printers, plotters, and magnetic tape. If microfilm records are required, the magnetic tape will be utilized for microfilm generation.

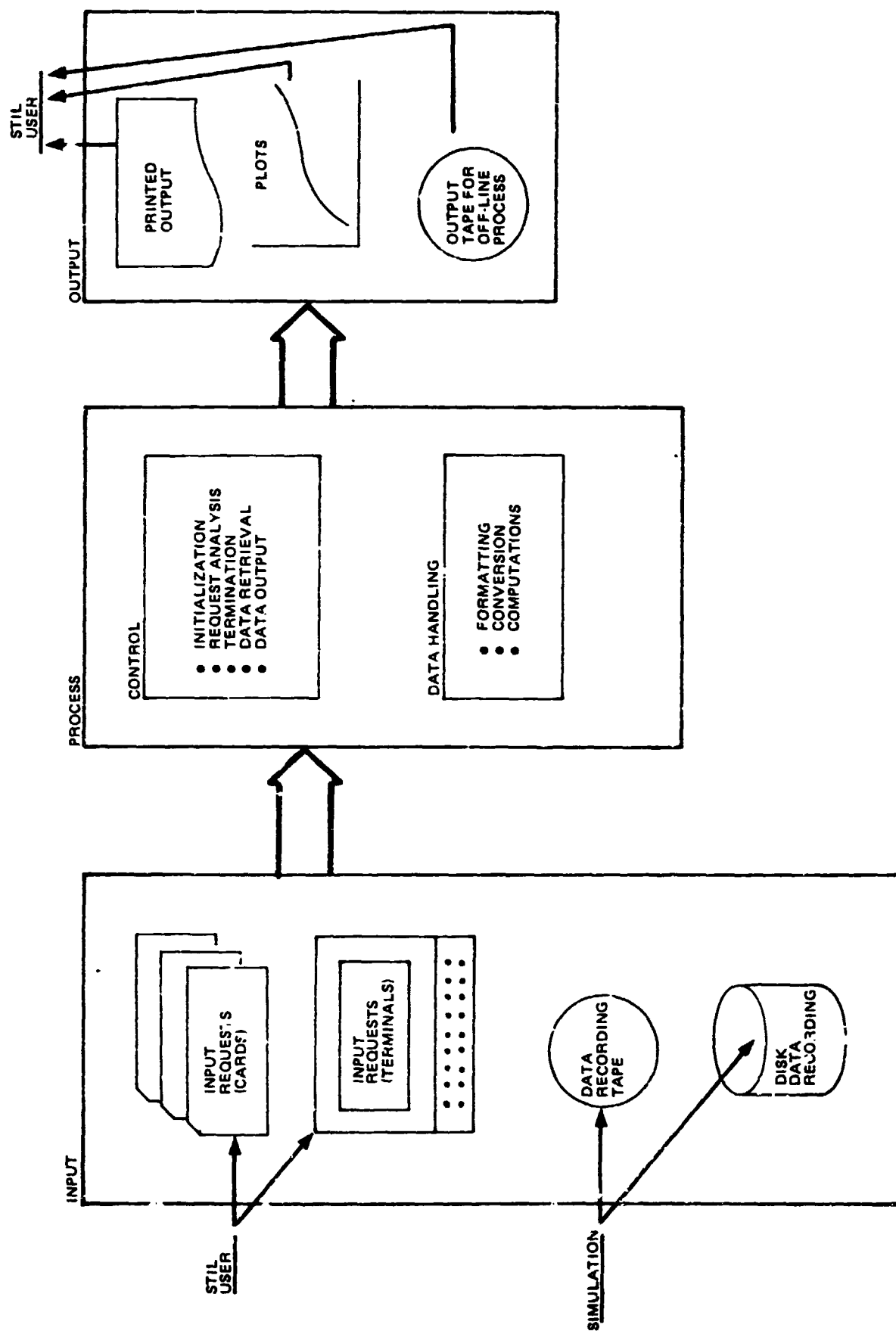


Figure 5-9. Functional Representation of Data Reduction Software

### User Aids

To support the STIL user in development and integration of CDMS computer and EGSE computer software, user aids must be provided. The user aids for batch mode will differ from those used in the realtime mode in that the host/CDMS hardware interface which must be supported in realtime does not exist in the batch mode. Specialized verification user aids will be required for automatically checking for proper use of programming standards and utilization of system interfaces.

The user aids which must be provided in the batch mode in support of software simulation have been included in the discussion of the simulation modes and will not be discussed in this section.

The principal user aid which must be provided to support batch users will be that of remote terminal interface with the host computer. For those users located remotely, the capabilities provided through terminals must include:

- Source data base maintenance
- Submission of batch jobs
- Use of system utilities

The output of remote job submissions will be mailed to the user or held for pickup, or transmitted to remote terminal printers.

### 5.3.3.3 Supportive Mode

To function in an efficient manner within the high development activity projected for the CDMS and EGSE software, the STIL must provide supportive resources to the STIL user. These supportive resources will consist of the host computer operating system, to maximize the utilization of the host computer resources, and software management tools to provide:

- Control over development activities
- Visibility into development status
- STIL utilization planning
- Configuration management of software

### Operating System

As has been established in previous sections, the STIL must support both realtime and batch processing environments. Because of the projected work volume to be accomplished on a daily basis, a multiprogramming operating system will be required. The Operating System (OS) concept envisioned for the STIL is shown in Figure 5-10. The basic operating system indicated within the figure has been assumed to be an operating system provided with the host computer.

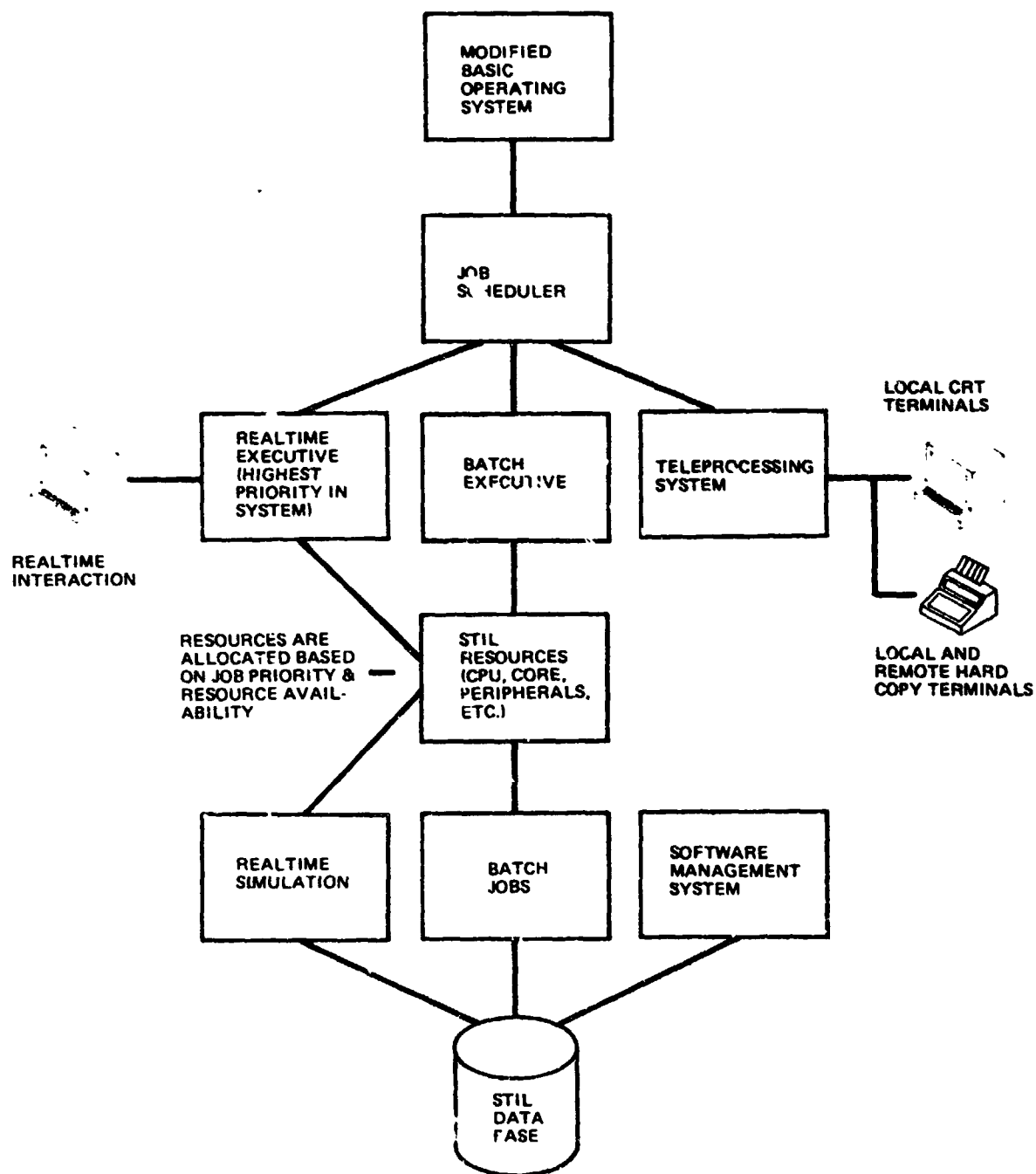


Figure 5-10. STIL Host Machine Operating System Concept

The Operating System must provide the following support services to users on the STIL:

- Job scheduling
- Supervisor services
- Data management
- Teleprocessing
- System utilities
- Host assemblers
- Host compilers
- Host linkage editors

These services, with their descriptions, are summarized in Table 5.2.

The realtime support requirements will necessitate modifications to the basic operating system. These modifications are considered to be unique to Spacelab and have been previously discussed in paragraph 5.3.3.1.

#### Software Management System

Because of the number of development activities underway simultaneously on the STIL, a software management system will be essential in providing an orderly flow. This software management system must utilize the STIL data base contents to provide a continuous status of overall STIL utilization and provide the STIL user with the resources needed to: (1) maintain software source statement files, (2) maintain configuration control over released software modules, packages, and sets, and (3) provide the capability to generate deliverable software sets under strict management controls. Previous experience on both the Saturn and Skylab programs indicates that the software management system should be a fully automated terminal-oriented data base management system.

As can be seen in Figure 5-11, the STIL data base will provide the means whereby software management can be achieved throughout the software development cycle. Working storage is provided within the data base for use by the development programmer during design, code, and debug activities and for use by the responsible programmer in module verification. Upon completion of module verification, the module will be placed under strict configuration control and placed into the program library.

**Table 5.2. Operating System Services**

OPERATING SYSTEM REQUIRED FEATURES	DESCRIPTION
Job Scheduling Services	Analyze input stream, allocate I/O devices, schedule jobs for execution, console operator communication interface.
Supervisor Services	Process interrupts, control CPU utilization, control memory allocation, and I/O supervision.
Data Management Services	Provides basic I/O access methods, allocates data sets on direct access devices, maintains a system catalog of data sets.
Teleprocessing Services	Controls terminal polling and addressing, receives and transmits messages and data between application and terminal users, controls remote job entry.
Utilities	Supports programs for the following:  TAPE DUMP/COPY/LABEL DISK DUMP/COPY/CREATE/RESTORE SYSTEM ERROR PROCESSING
Assembler	A host machine assembler with extensive macro features.
Compiler	A HOL compiler such as PL1, Fortran.
Linkage Editor	Combines program modules into a loadable form for execution on the host machine.



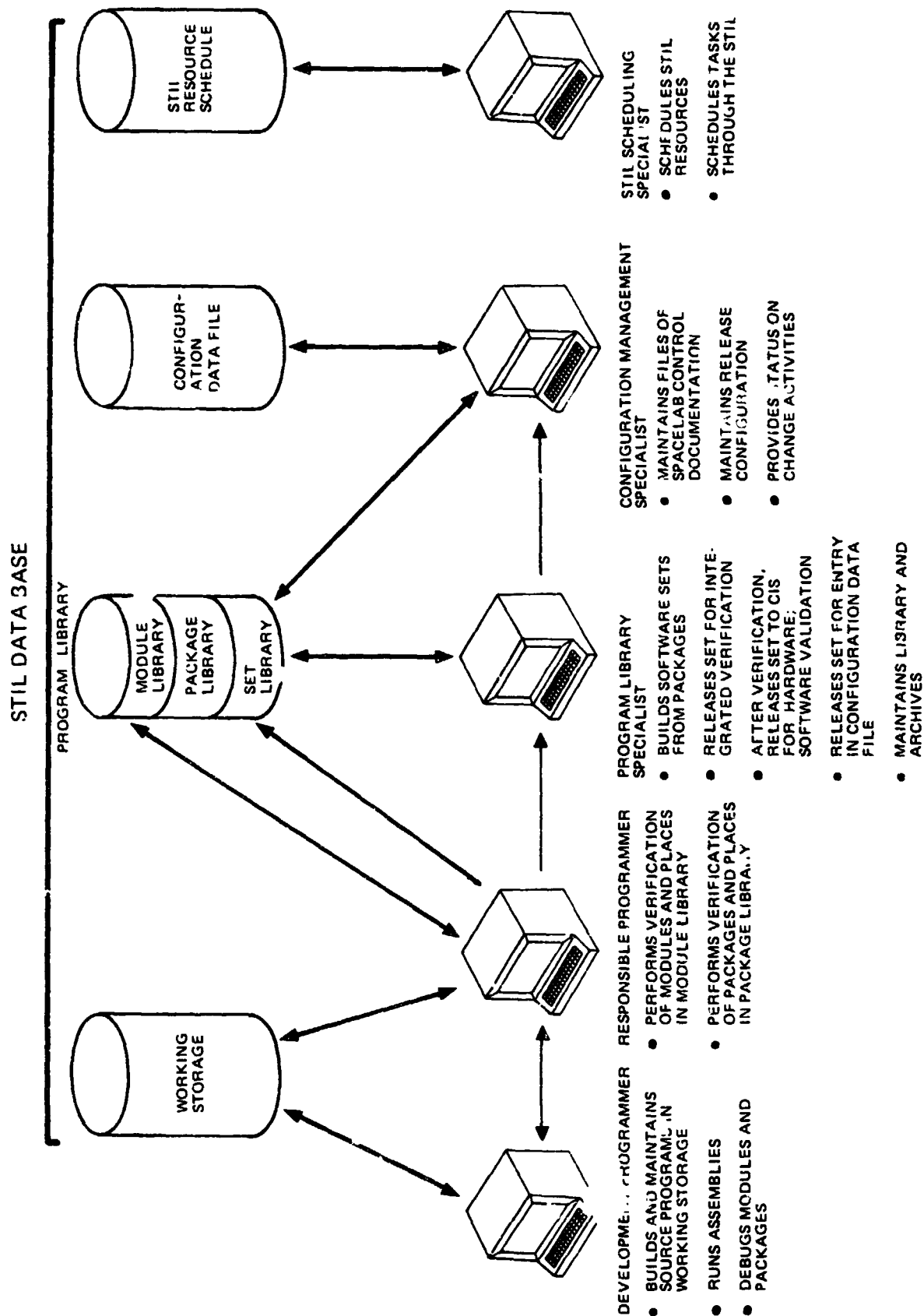


Figure 5-11 Proposed Software Management System Flow

The responsible programmer will utilize the modules within the program library to construct software packages needed to support particular Spacelab experiments. These packages are verified and then placed into the program library under configuration control.

The packages are utilized to construct software sets in support of Spacelab missions. The sets are verified prior to release to the CIS for hardware/software validation. Upon completion of verification, the sets, with associated documentation, are placed into the program library under configuration control. The sets will remain in the on-line data base for 60 days and then will be sent to archives; however, a catalog of the contents of each set and the location of associated packages and modules will be maintained within the on-line data base. This capability will allow the reconstruction of a set for application to identical Spacelab missions.

The software management system also will provide the capability to plan STIL resource utilization. This capability will be essential to efficient use of the STIL in an environment in which up to 18 sets could be undergoing development simultaneously.

The following paragraphs will discuss the software systems required of the STIL in order to support the software management system requirements.

The source data management system will provide the STIL user with the capability to store and maintain source statements within partitioned data sets of the STIL data base. The source data for each version and level of a module will be assigned a unique identifier to allow updates from either terminals or background batch update. Each source module will have a directory entry which will uniquely identify the module by date, version number, and revision level and will maintain a change history for the module. Within the software management system concept, only the development programmer or responsible programmer will update source statements during development and module verification. No modification of source statements will be allowed within modules residing in the program library or configuration data file.

The utilization of the source data management system is shown in Figure 5-12.

The configuration management system will provide the capability to maintain and track the composition and documentation of all released program sets. In addition, the configuration management system will maintain data files for tracking of problem reports and change requests written against Spacelab CDMS and EGSE software. Included within the configuration data file (CDF), utilized by the configuration management system software, will be source, object, and documentation for all modules, simulators, and other software utilized within the STIL in support of software development.

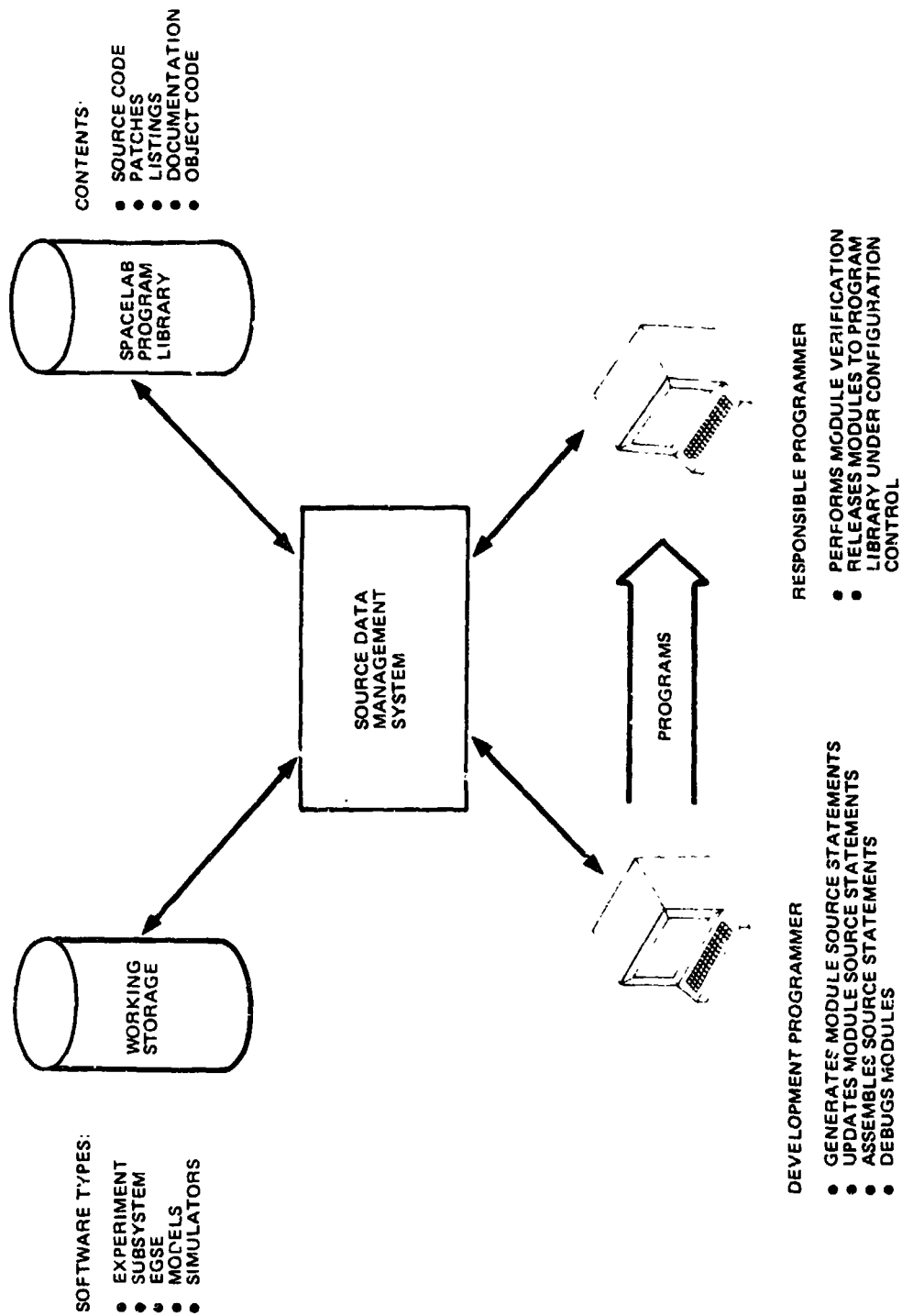


Figure 5-12. Source Data Management

The use of the configuration management system is pictorially represented in Figure 5-13.

The automatic release system will be utilized to automatically build software sets from the program library of the STIL data base. Through input from the program library specialist, the software packages, with associated documentation, will be combined into sets for verification. Upon completion of verification, the set will be released for validation and will be entered into the configuration data file.

The use of the automatic release system is shown in Figure 5-14.

The STIL scheduling system must be capable of collecting all the individual task requirements, such as completion dates and STIL resources, needed to complete the task. The system will utilize this data to produce a STIL utilization schedule. This schedule will then be compared with the actual STIL resources available and any incompatibilities noted for management attention. As tasks are in process, estimated or actual completion dates and resource utilization requirements must be entered into the system. These new data elements will be compared with existing data and any current or future incompatibilities reported.

Schedule and resource information must be stored in a STIL schedule data file which will be a part of the overall STIL data base. Schedule changes, additions, deletions, completion dates and changes to resource baselines must be entered via terminals. Status reports and revised schedules can be either retrieved on-line via terminals or obtained via a background task run in the batch mode.

The scheduling system is shown in Figure 5-15.

#### 5.3.3.4 STIL Data Base Definition

The STIL data base size will be the significant factor in determining the capacity of on-line (disk) storage capacity which must be fully provided. In addition, the requirement to maintain a backup capability for the data base will assist in determination of record keeping/magnetic tape storage requirements in support of the STIL. To establish a preliminary estimate of the physical size of the STIL data base, a summary of data base elements has been accomplished. The sizes of these elements were developed through analysis or were based on similar functions existing within Saturn and Skylab software development data bases. As can be seen in the summary, Table 5.3, the data base size approaches 1.3 billion bytes of data.

The following sizing assumptions relating to Table 5.3 were made in establishing the STIL data base size:

1. Source statements, load modules, and listings will be retained for all development support and host software dedicated to Skylab released systems.

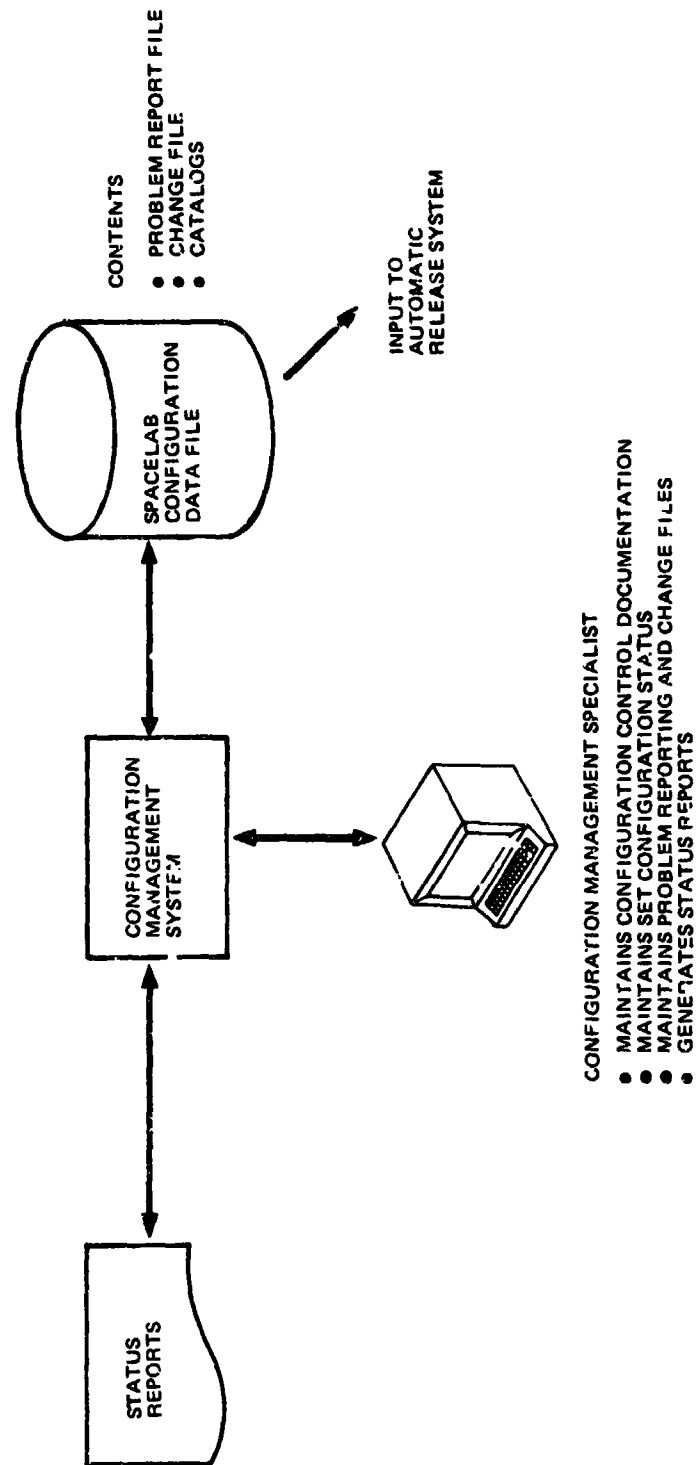


Figure E-1.2 Configuration Management System

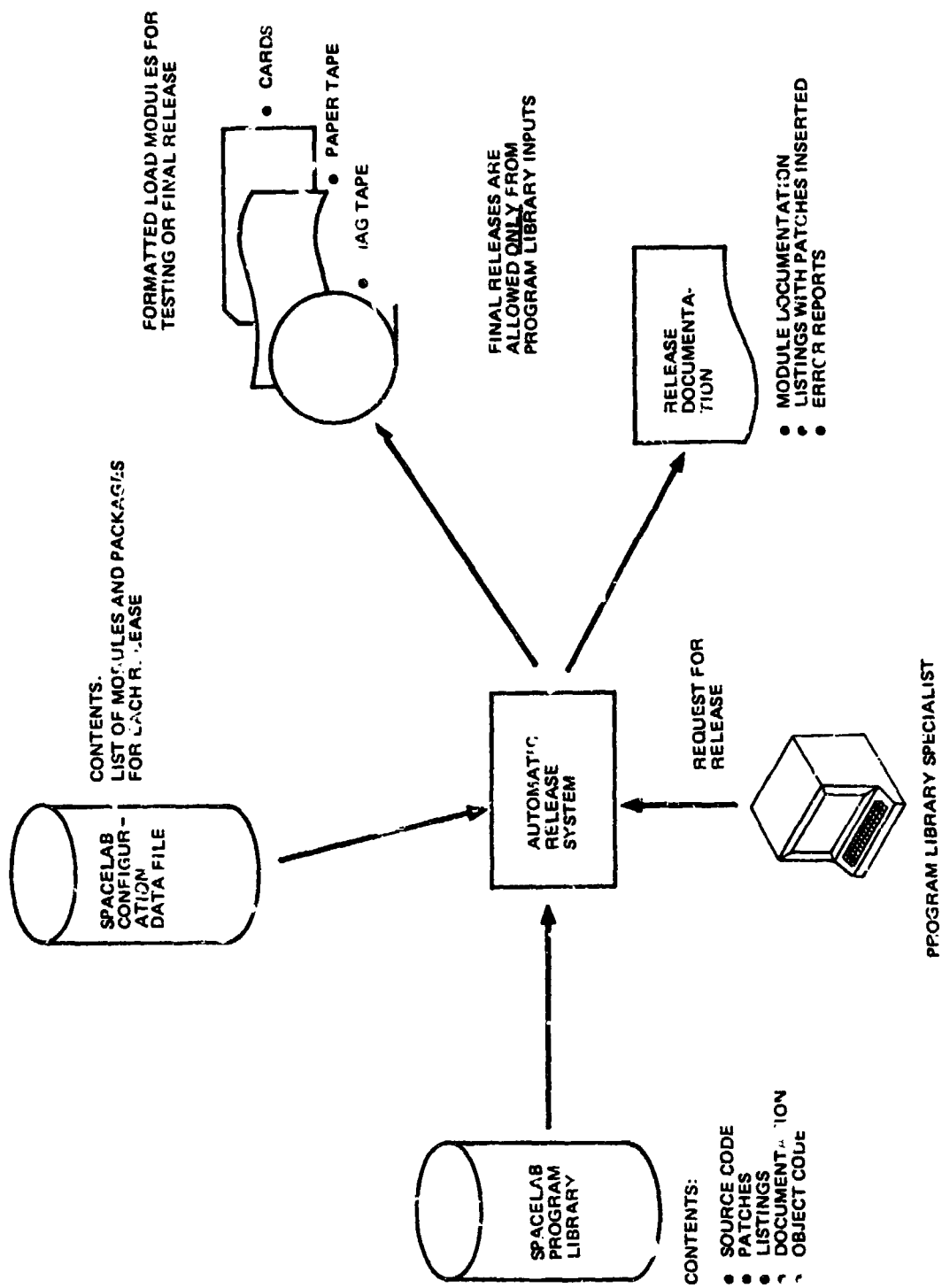


Figure 5-14. Automatic Release System

In support of integration Levels I, II, and III, the Test and Checkout software must provide the varying capabilities necessary to perform the required testing. Level IV integration is considered a responsibility of the experiment PI during manufacture and is not considered a portion of the Test and Checkout software concept.

In addition to supporting the above integration levels, the Test and Checkout software must perform on-orbit testing of the Spacelab subsystem and experiments. Implicit within the support requirement is the participation of the Shuttle Orbiter and Payload Operations Center.

CONTENTS:

- TASK AND RESOURCE REQUIREMENTS
- SCHEDULE CHANGES
- COMPLETION DATE
- STATUS REPORTS
- RESOURCE BASELINE

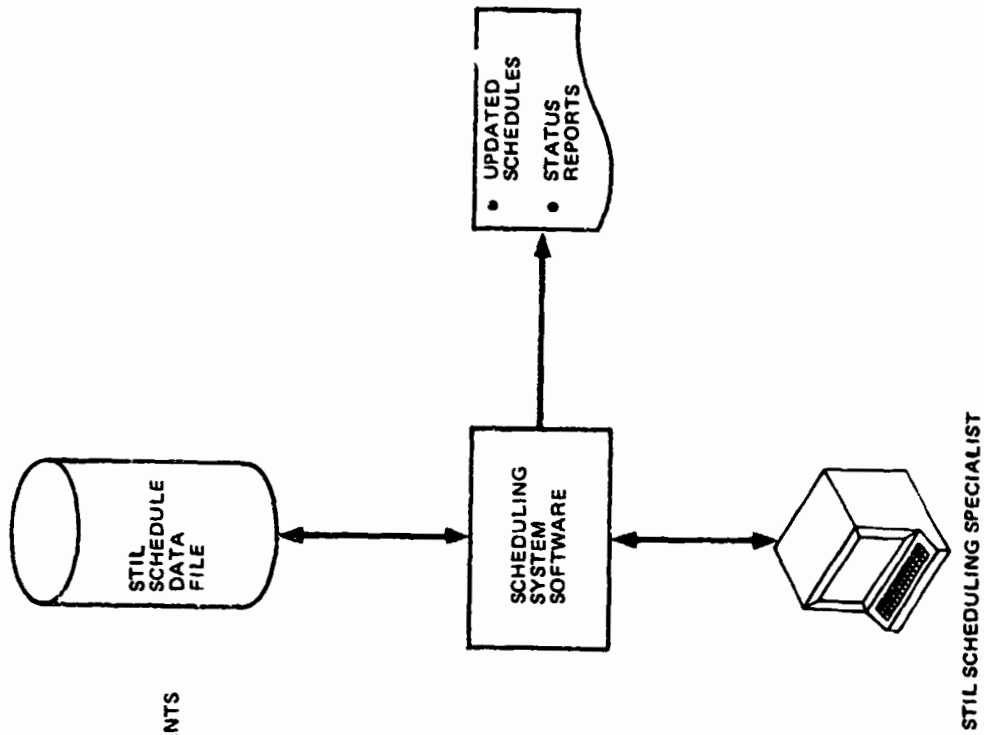


Figure 5-15. St. acelab Software Scheduling System



Table 5.3. Preliminary STIL Data Baseline

	HOL STATE- MENTS	LOAD MODULE (BYTES)	LINES OF LISTING	SIZING BASIS
<b>DEVELOPMENT ACTIVITY</b>				
Experiment Flight Applications Software	7.2K	144.0K	21.6K	Analysis
Experiment Ground Support Software	3.8K	76.8K	11.4K	ESRO
Subsystem Flight Software	2.8K	56.0K	8.4K	ESRO
Subsystem Ground Checkout Software	11.6K	220.0K	33.0K	ESRO
EGSE Ground Checkout Software	13.6K	272.0K	40.8K	ESRO
<b>SIMULATIONS</b>				
Real Time Simulation	45.0K	900.0K	135.0K	Saturn/Skylab
ICS	43.3K	876.0K	132.0K	Saturn/Skylab
Functional Simulation	27.2K	544.0K	81.6K	Saturn/Skylab
Design Analysis	10.4K	208.0K	31.2K	Saturn/Skylab
Data Reduction	15.0K	300.0K	45.0K	Saturn/Skylab
<b>SUBTOTALS</b>	<b>179.3K</b>	<b>3.596M</b>	<b>540.0K</b>	
<b>18 PACKAGES TOTAL</b>	<b>3.23M</b>	<b>64.74M</b>	<b>9.72M</b>	
<b>BYTES OF STORAGE</b>	<b>258.2M</b>	<b>64.74M</b>	<b>777.6M</b>	
<b>STIL FACILITY SUPPORT</b>				
Operating System (Including Real Time)		300.0K		OS 360 Version 21
CDMS Computer Operating System	4.0K	80.0K	12.0K	Analysis
EGSE Operating System	4.0K	80.0K	12.0K	Analysis
Released Spacelab Sets (10)	364.0K	7.69M	1.152M	
HOST Compiler		480.0K		Fortran and PL/I
HOST Macro-Assembler		80.0K		360 Assembler
HOST Linkage Editor		122.0K		360 Linkage Editor
CDMS HOL Compiler	11.0K	220.0K	33.0K	GOAL Compiler
CDMS Assembler	10.0K	200.0K	30.0K	Skylab Onboard Assembler
CDMS Linkage Editor	14.5K	290.0K	43.5K	Skylab Linkage Editor
<b>SIMULATIONS FOR RELEASED SPACELAB SOFTWARE (10)</b>				
Real Time		9.0M		
ICS		8.76M		
Functional		5.44M		
Design Analysis		2.08M		
<b>SOFTWARE MANAGEMENT</b>				
Source Maintenance	1.5K	30.0K	4.5K	OS 360
Configuration Management	9.0K	180.0K	27.0K	Saturn/Skylab
Automatic Release	15.0K	300.0K	45.0K	Saturn/Skylab
STIL Scheduling	10.0K	200.0K	30.0K	Analysis
<b>SUBTOTALS</b>	<b>463.0K</b>	<b>35.5M</b>	<b>1.78M</b>	
<b>BYTES OF STORAGE</b>	<b>37.0M</b>	<b>35.5M</b>	<b>142.0M</b>	
<b>OVERALL TOTAL (BYTES)</b>	<b>295.2M</b>	<b>100.0M</b>	<b>919.6M</b>	
<b>OVERALL TOTAL - 1.315 BILLION BYTES STORAGE REQUIRED</b>				

2. A source statement will generate an average of five machine language instructions upon compilation. Storage of each statement will require 80 bytes (in compressed form).
3. Each machine language instruction will utilize four bytes of storage.
4. Each line of listing requires 80 bytes of storage (in compressed form). Each source statement generates approximately three lines of listings.
5. A maximum of 18 sets undergoing development must be simultaneously supported. For the preliminary sizing, it was assumed that all elements listed under development support would be required for all 18 sets.
6. Those elements listed under host facility support will remain stable throughout the development cycle.
7. A copy of the released software sets and supportive simulations will be maintained within the on-line data base for 60 days after the mission has been flown for post-mission support activity. This results in 10 sets being retained within the data base at all times (5 CDMS and 5 EGSE).
8. The source statements and listings of host-provided software and released simulators would not be maintained within the data base.

#### 5.3.3.5 STIL Load Requirements

To maintain the software and hardware capabilities of the STIL, a series of maintenance runs will be required on a daily basis. The areas of the system requiring this daily activity with the corresponding runs per day are summarized in Table 5.4. Also shown in the table are the reasons for the runs.

**Table 5.4. STIL Load Requirements**

TYPE OF RUN	REASONS	RUNS/DAY
SYSTEM MAINTENANCE	<ul style="list-style-type: none"> <li>● Saving of Critical System Data on Magnetic Tape</li> <li>● Maintenance of Catalogs</li> <li>● Procedure/Job Library Maintenance</li> <li>● Debug of Suspected System Problems</li> <li>● Updates to Operating System</li> <li>● System Statistics Gathering</li> </ul>	<p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p>
SOFTWARE MANAGEMENT	<ul style="list-style-type: none"> <li>● Generation of Software Status Reports (Changes, Problems, Testing)</li> </ul>	1
STIL SOFTWARE SCHEDULING	<ul style="list-style-type: none"> <li>● Generation of Overall STIL Utilization Status</li> </ul>	1
TOTAL		8

## 5.4 STIL MODELING ANALYSIS

### 5.4.1 THEME

A General Purpose Simulation System (GPSS) program was developed to assist in bounding the CPU computational and memory requirements of the STIL host computer complex. Inputs to the model were established in the process of performing analyses of Experiment Flight Applications software, Test and Checkout software, and STIL Supportive software.

### 5.4.2 CONCLUSIONS

As a result of the STIL modeling analysis, the following conclusions have been established:

- The preliminary STIL modeling has indicated that a CPU power of 3 MIPS and a memory capacity of 3 million bytes will be required of the host computer complex and is the optimum configuration from a job throughput standpoint and future growth potential.
- Detailed modeling of the STIL must be performed prior to selection of the final computer configuration.
- The major burden on the STIL will be the realtime simulation requirement because of its large memory requirements.

### 5.4.3 DISCUSSION

Within the Spacelab software test and integration task, the STIL modeling analysis provided the means for compiling all STIL requirements, which the host computer must support, and developing the required computer characteristics. The modeling analysis activity will be discussed in the following manner:

- Model description
- Model input definition
- Model utilization
- Model results

#### 5.4.3.1 Model Description

The STIL model was developed in the General Purpose Simulation System (GPSS) language and provided the capability to simulate execution of both realtime and background processing tasks. The description of the flow of tasks within the model is shown in Figure 5-16.

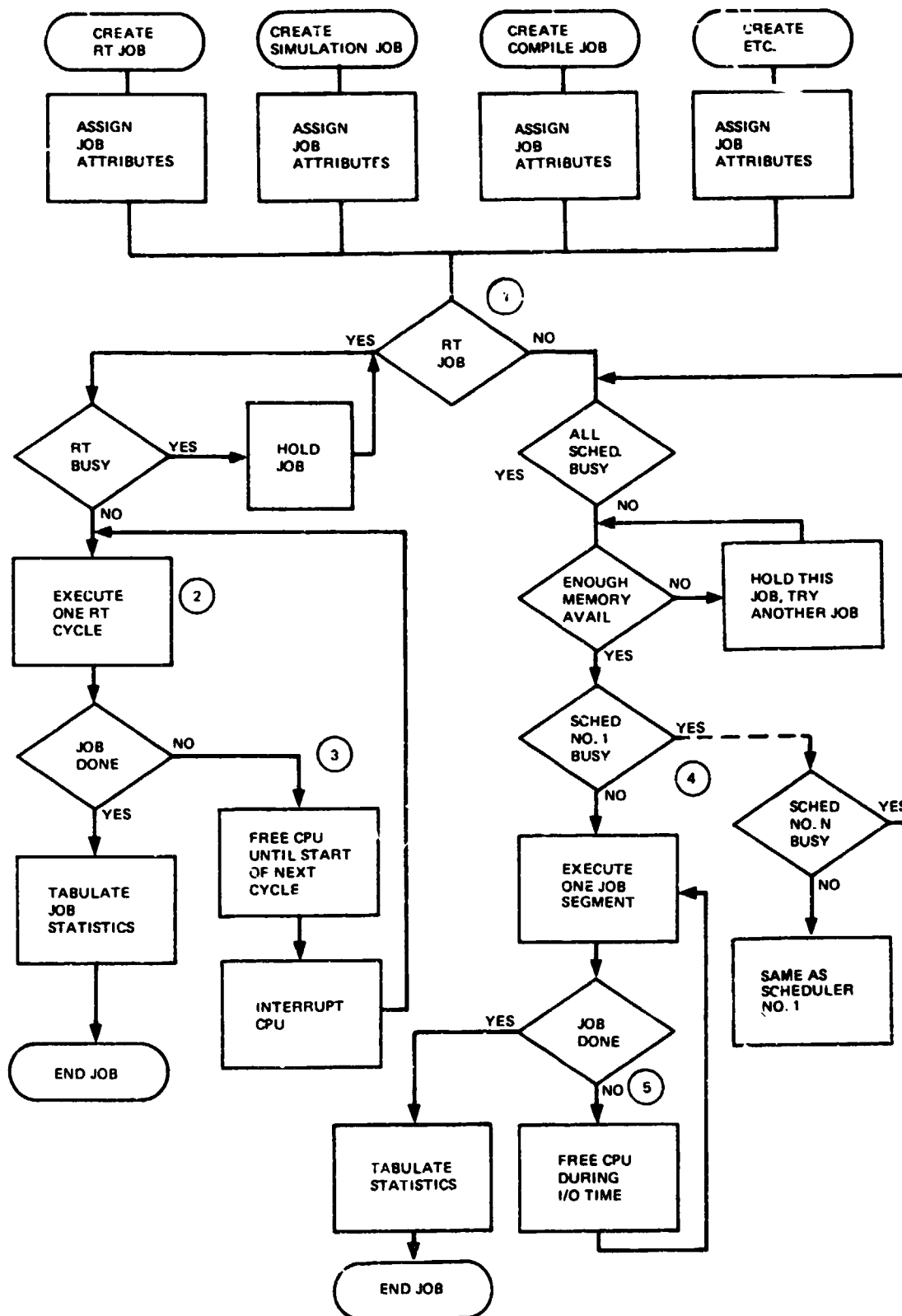


Figure 5-16. Preliminary STIL Configuration Model Task Flow

As can be seen in the figure, tasks are categorized according to task class and assigned attributes such as runs/day, time per run, and memory requirements. The model utilized these attributes to simulate the flow of tasks within the host computer. The model assumed an unlimited input/output capability with standard 360/70 I/O "wait" times.

#### Realtime Task Simulation

At flow block number 1, a decision is made for type of job. If a realtime job, flow will proceed down the left side of the figure. If side of the flow figure. Only a single realtime job may be executed at a time. If no realtime job is operating, a new realtime job will start into execution at flow block number 2 immediately upon entering the model. The realtime job will run on a high priority, cyclic basis on precise time intervals to simulate a processing interface with the CDMS. The cycle includes processing time and I/O wait time. The wait time is represented by flow block number 3. During this wait time, the CPU is available for any of the other jobs in the model.

At the precise time for the start of the next realtime cycle, the model simulates a priority interrupt which takes the CPU away from a lower priority job and performs a realtime processing cycle. When the realtime job again goes into the wait condition at flow block number 3, the interrupted job continues from its point of interruption.

The realtime job continues cyclic execution until the specified execution time of the job has been reached, at which time the job statistics are tabulated and the job is removed from the model.

#### Background Task Simulation

No realtime jobs are processed by any of the background (batch/support) schedulers. The background jobs are gated into a executing state by the availability of memory and a scheduler. If these conditions are satisfied, the job begins execution at flow block number 4. The background jobs also execute in cycles similar to the realtime job. Flow block number 5 simulates the I/O wait time required between CPU processing cycles of the job.

Several background jobs may be in execution at the same time. The number of jobs in execution at any given time is limited by the availability of memory and a job scheduler. The number of schedulers was a model parameter and was adjusted to maximize memory utilization. The background jobs must all share the CPU according to priorities and, in effect, share the CPU resources left over from the realtime jobs. This sharing is accomplished by executing a cycle for a job under scheduler number 2 (not shown) and so on for each scheduler that has an active job. A background job continues cyclic processing until the specified CPU time of the task has been reached. Statistics on the job are then tabulated and the job removed from the model.

#### 5.4.3.2 Model Input Definition

In order to utilize the model, the tasks to be executed and the attributes of the tasks were compiled from the STIL load factors established in the Experiment Flight Applications software, Test and Checkout software, and STIL Operational Analyses. The compilation of this total load is shown in Tables 5.5 and 5.6. Table 5.5 indicates the frequency of task execution with each task. The attributes of the tasks (core, cycle/job, CPU time, run time) were obtained from analysis of realtime simulation data accumulated during the Skylab program and from statistics gathered from a typical large scale computer center for background processing, and are found in Table 5.6.

#### 5.4.3.3 Model Utilization

Having established the task frequencies and resource attributes for the daily STIL load, a series of modeling cases were executed. The objective of these cases was to optimize the CPU power and memory size combination in order to effectively handle the daily STIL job load. A description of each case and its results are detailed in the following paragraphs:

##### Case 1:

##### Description

Holding memory constant while varying CPU capability.

##### Discussion

The first case iteration consists of holding memory constant at 2 million bytes while the CPU power was varied from 1 MIPS (million instructions per second) to 5 MIPS. From these results one can conclude that 1 MIPS machine cannot handle the processing load. The 2 MIPS machine came close, but memory size seemed to be one of the limiting factors. The 3, 4 and 5 MIPS machines can handle the loads with jobs being processed, as they are initiated, with little or no wait time.

##### Results

Any increase in computing power after the system has become I/O bound has little or not effect on system throughput (reference Figure 5-17) using the job mix expected in the STIL.

*Table 5.5 STIL Job Frequencies*

JOB TYPE	RUNS/DAY	FREQUENCY PER 16-HOUR DAY
ASSEMBLY	7	$137 \pm 10$ Min.
COMPILE	59	$16 \pm 5$ Min.
LINK EDIT	8	$120 \pm 10$ Min.
UTILITIES & SYSTEM MAINTENANCE	8	$120 \pm 20$ Min.
SIMULATIONS		
• Functional		
• ICS		
• Data Analysis	77	$12 \pm 10$ Min.
REALTIME TASKS		
• Interactive		
• CDMS Hardware Sim		
• Models		
• Data Reduction		
• User Aids	31	$30 \pm 10$ Min.
DATA REDUCTION	91	$10 \pm 5$ Min.
MODEL (EXPERIMENT) MAINTENANCE	22	$45 \pm 10$ Min.



Table 5.6. STIL Model Input Data Summary

JOB TYPES	CORE REQ'T (K)	CYCLES (1) PER JOB	TIME (2) PER CYCLE		TOTAL CPU TIME (Sec)	TOTAL RUN TIME	JOB (3) FREQUENCY (Min)	TOTAL RUNS/DAY
			CPU (Sec)	I/O (Sec)				
ASSEMBLIES	125	8	1	14	8	2	137 ± 10	9
COMPILES	200	9	2	25	18	4	16 ± 5	75
LINK EDITS	130	3	1	40	3	2	120 ± 10	10
UTILITIES & SYSTEM MAINTENANCE	70	5	1	60	5	5	120 ± 20	23
SIMULATION	340	120	5	1	600	12	12 ± 10	89
Functional								
Interpretive Computer Sim								
Design Analysis								
REALTIME TASKS	1000	90	2	1	180	5	30 ± 10	37
Interactive								
CDMS F/W Sim.								
Models								
Data Reduction								
User Aids								
DATA REDUCTION	150	120	4	1	480	10	10 ± 5	105
EXPERIMENT MODEL MAINTENANCE	40	1	1	1	1	2 Sec.	45 ± 10	22
WRITER TASK (4)	60	Continuous	1	4	—	—	Continuous	
TOTAL RUNS/DAY								350

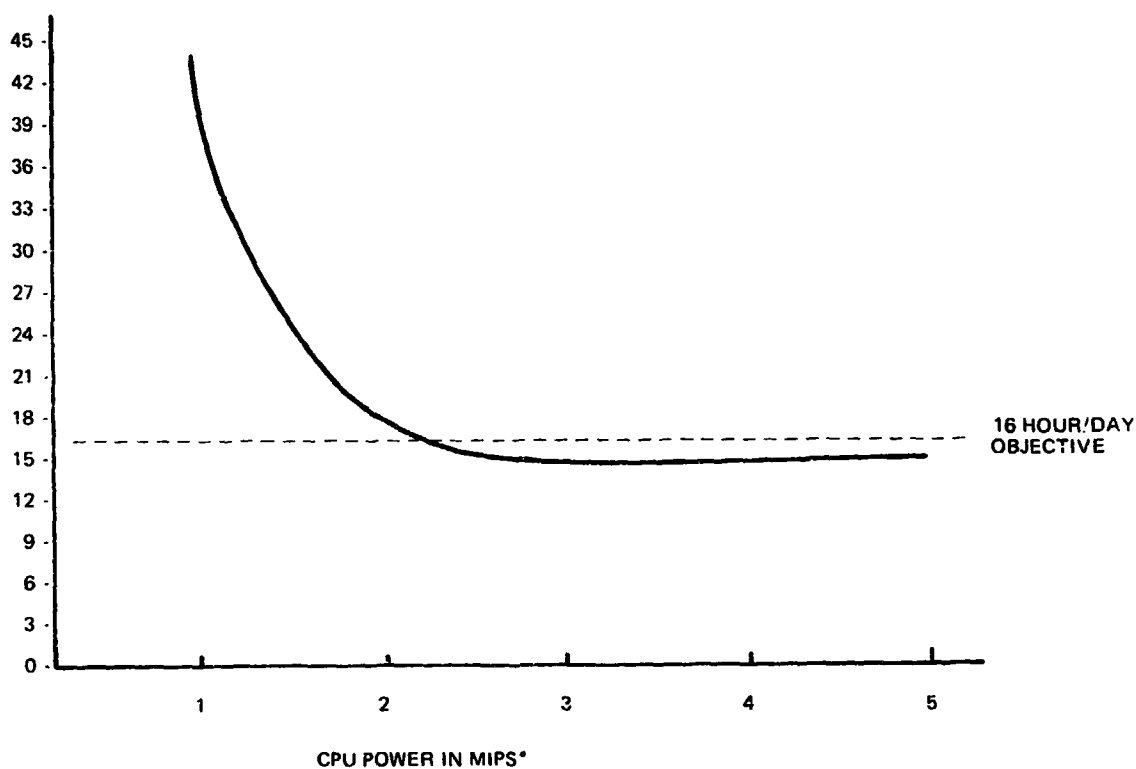
- (1) ITERATIONS OF EXECUTING CPU PROCESSING AND I/O WAITS.
- (2) REPRESENTATIVE JOBS WERE ANALYZED FOR CPU TO I/O RATIOS AND TIMES.
- (3) RATE AT WHICH THESE TYPE JOBS ENTER THE STIL.
- (4) SIMULATES ALL THE I/O OVERHEAD IN THE SYSTEM (PRINTERS, DISKS, ETC')

**RUN DESCRIPTION:**

- MEMORY HELD CONSTANT AT 2 MILLION BYTES
- I/O CAPABILITIES HELD CONSTANT
- REALTIME TASKS REQUIRE 1 MILLION BYTES OF MEMORY
- VARIED CPU POWER FROM 1 TO 5 MIPS \*
- JOB LOAD HELD CONSTANT

**CONCLUSION:**

UPON ACHIEVING THE CPU CAPABILITY TO HANDLE THE COMPUTATIONAL BURDEN, ADDITIONAL CPU CAPABILITY DOES NOT INCREASE SYSTEM THROUGHPUT. RATHER THAN INCREASING THROUGHPUT, THE AMOUNT OF WAIT TIME IS INCREASED.



\*MIPS - Million Instructions Per Second

*Figure 5-17. Throughput as a Function of CPU Capability*

#### Case 2:

##### Description

Holding CPU constant while varying memory capability.

##### Discussion

The second case iteration consists of holding the CPU power at 1 MIPS and varying the core size from 1600K to 4000K. Even with the varying core size, the 1 MIPS machine could not handle the processing load effectively.

##### Results

Inadequate CPU powers could not effectively handle the STIL job load requirement even with increases in memory size (see Figure 5-18).

#### Case 3:

##### Description

Based on results of Cases 1 and 2, select more reasonable host computer characteristics for CPU and vary memory.

##### Discussion

The third case iteration consisted of holding the CPU power constant at 2 MIPS and varying core size from 1600K to 4000K. The data gathered from this exercise indicated that the 2 MIPS CPU with any combination of memory size remained CPU bound and could not process the job load. It was therefore concluded that when a given CPU power becomes CPU bound, increases in memory size do not alleviate the problem.

##### Results

Inadequate CPU powers could not effectively handle the STIL job load requirements even with increases in memory size (reference Figure 5-19).

#### Case 4:

##### Description

Select more powerful CPU capabilities and varying memory capability.

##### Discussion

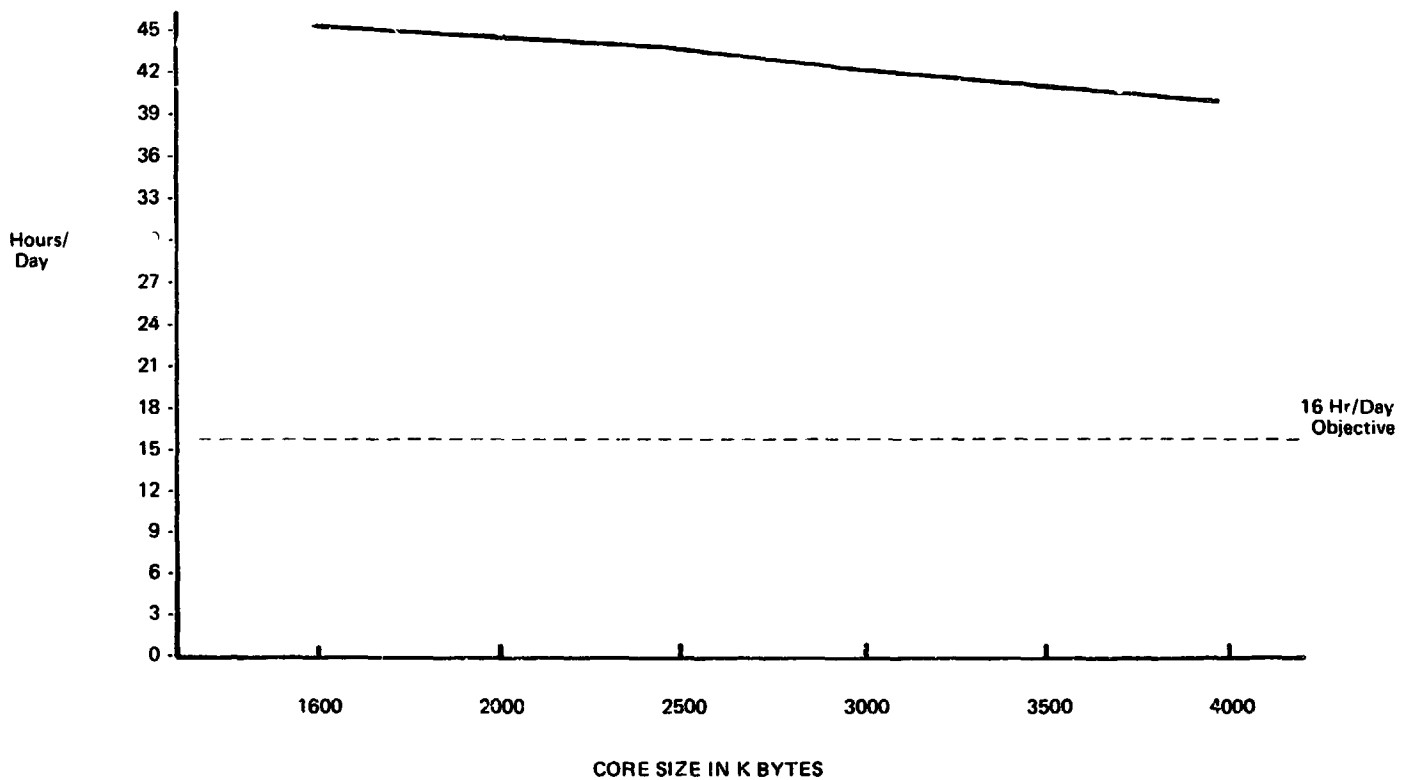
The fourth case iteration consisted of holding the CPU power constant at 3, 4, or 5 MIPS while varying the core size from 1600K to 4000K for each CPU capability. All three CPU powers could handle the job load within the

**RUN DESCRIPTION:**

- CPU POWER HELD CONSTANT AT 1 MIPS
- I/O CAPABILITIES HELD CONSTANT
- REALTIME TASKS REQUIRE 1 MILLION BYTES OF MEMORY
- VARIED CORE SIZE FROM 1600K to 4000 K.
- JOB LOAD HELD CONSTANT

**CONCLUSION:**

INCREASING CORE SIZE AT 1 MIPS CPU POWER  
HAS NO APPRECIABLE EFFECT ON THROUGHPUT



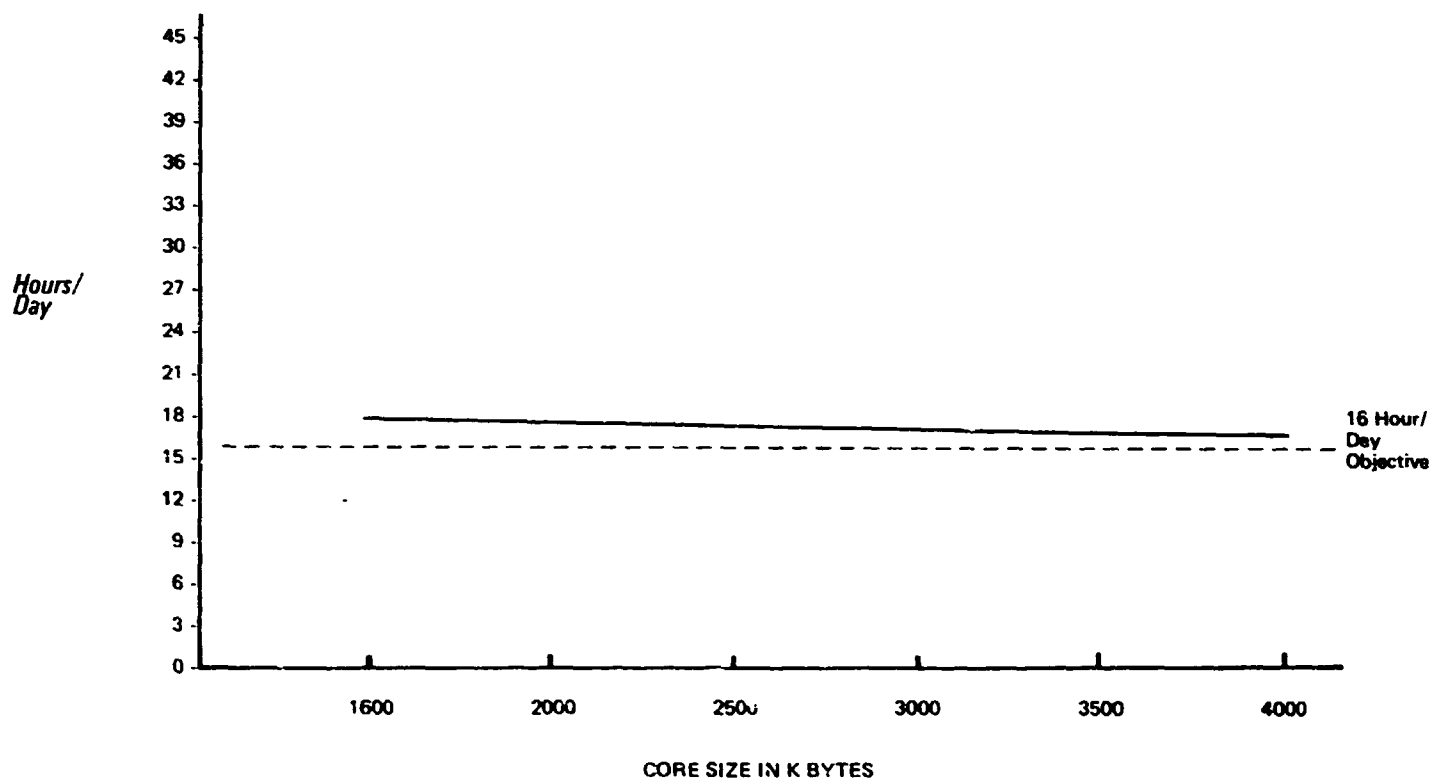
*Figure 5-18. Throughput as a Function of Core Memory for 1 MIPS CPU*

**RUN DESCRIPTION**

- CPU POWER HELD CONSTANT AT 2 MIPS
- I/O CAPABILITIES HELD CONSTANT
- REALTIME TASKS REQUIRE 1 MILLION BYTES OF MEMORY
- VARIED CORE SIZE FROM 1600K TO 4000K
- JOB LOAD HELD CONSTANT

**CONCLUSION:**

FOR 2 MIPS CPU, INCREASING MEMORY  
CAPACITY HAS LITTLE EFFECT ON THROUGHPUT.



*Figure 5-19. Marginal CPU Capability Relative to Memory Allocation*

time objective. After reaching this point, increased memory was of no appreciable help. This condition of leveling off is due to the characteristics of the input job stream, i.e., jobs are initiated over the entire 16 hour period. The three CPU powers fared identically as far as processing required to meet the time objective, but differed in CPU utilization. The 3 MIPS model had a CPU utilization of 77%, the 4 MIPS a utilization of 63%, and the 5 MIPS a utilization of only 54%; thus leading to the conclusion that the higher powered CPUs stay in a wait or idle state much of the time when the system can handle the job load.

### Results

With the adequate CPU powers (3, 4, and 5 MIPS) while varying the memory size, there is a point at which the system ceases to be effective and further increases in memory size have no effect on system throughput (reference Figure 5-20).

### Case 5:

#### Description

Determine growth potential for those combinations satisfying load requirements.

#### Discussion

The fifth case iteration consisted of holding the CPU power constant at 1, 2, 3, 4, and 5 MIPS respectively, varying core size from 1600K to 4000K per each CPU power and placing a time constraint on each combination to determine the growth potential, if any, for each combination. The base time constraint was the job load required during a 16-hour day. For a 33% growth potential the same number of jobs required during the 16-hour day must be processed in 12 hours. A 60% growth potential requires that the same number of jobs required for a 16-hour day be processed in 10 hours. A 100% growth potential requires that the 16-hour baseline load be processed in 8 hours.

### Results

A growth potential chart was derived from applying the same load requirements for a 16-hour day and attempting to efficiently handle this load in 8, 10, and 12 hours while varying the CPU and memory size requirements (reference Figure 5-21). Based on a given growth factor, a combination of a CPU and a memory size may be determined from this chart.

**RUN DESCRIPTION:**

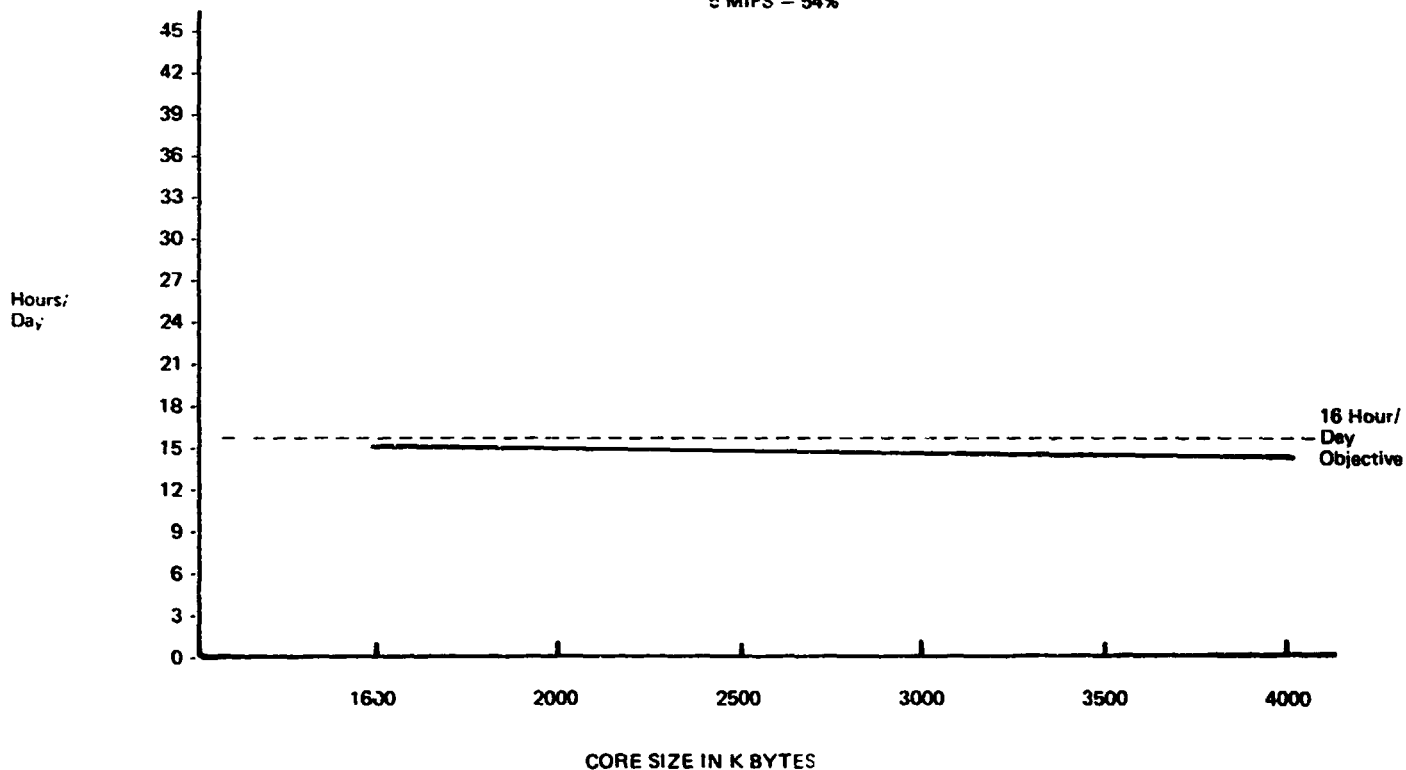
- CPU POWER HELD CONSTANT AT 3, 4 AND 5 MIPS
- I/O CAPABILITIES HELD CONSTANT
- REALTIME TASKS REQUIRE 1 MILLION BYTES OF MEMORY
- VARIED CORE SIZE FROM 1600K TO 4000K
- JOB LOAD HELD CONSTANT

**CONCLUSION:**

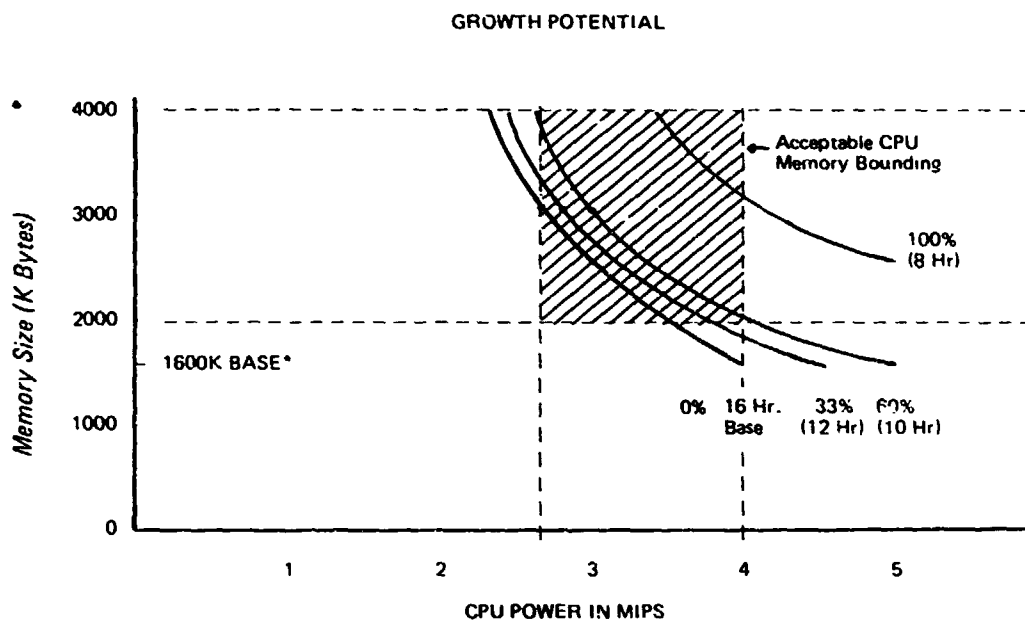
THESE 3 CPU POWERS WERE CAPABLE OF HANDLING THE LOAD WITHIN THE TIME OBJECTIVE. HOWEVER, BECAUSE OF THE JOB STREAM CHARACTERISTICS, INCREASING CPU POWER HAD LITTLE EFFECT ON THROUGHPUT. RATHER THAN INCREASING THROUGHPUT, WAIT TIME INCREASES.

**CPU UTILIZATION:**

3 MIPS - 77%  
4 MIPS - 63%  
5 MIPS - 54%



*Figure 5-20. Impact of Insufficient I/O Capability*



\* MINIMUM MEMORY SIZE TO SUPPORT: 300K OPERATING SYSTEM  
1000K REALTIME APPLICATIONS  
300K BATCH JOBS

This chart shows an optimal CPU/Memory combination to be a 3 MIPS CPU power and 3 million byte memory with a growth potential of 60%.

*Figure 5-21. Growth Computer Profiles*



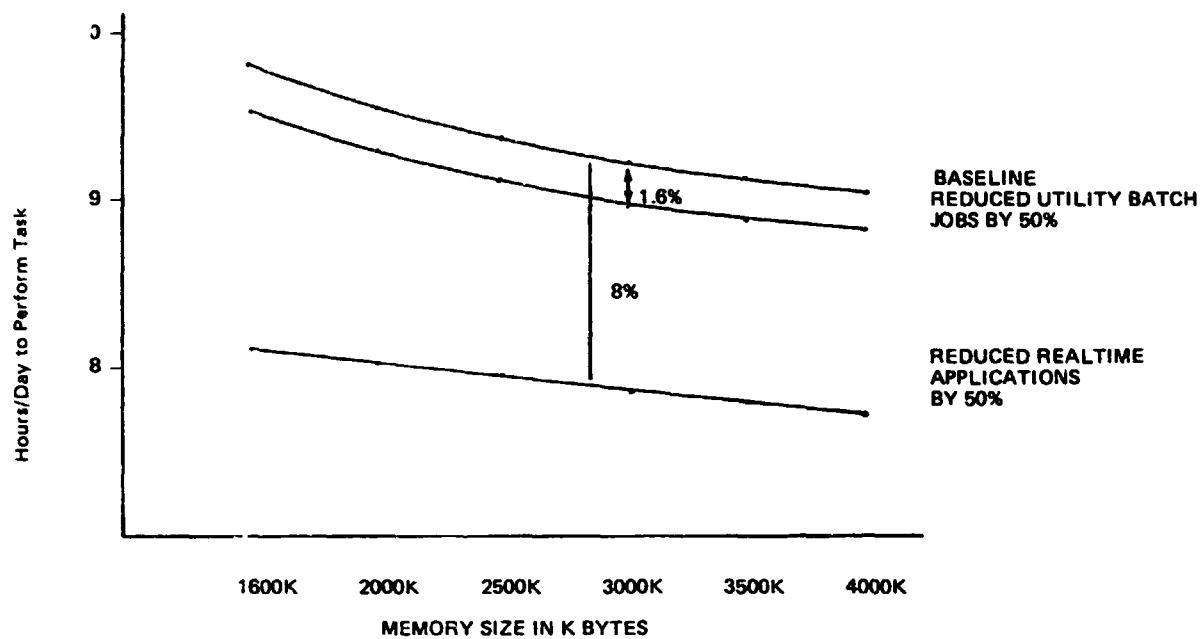


Figure 5-22. Sensitivity Analysis

#### Case 6:

##### Description

Determine the effects of reduced job requirements for realtime applications and utility background task respectively.

##### Discussion

The sixth case iteration consisted of performing a sensitivity analysis on the model by first reducing the number of required utility background tasks (assembles, compile, link edits, etc.) by 50%, and then by reducing the number of required realtime tasks by a factor of 50%.

The test baseline consisted of a CPU of 3 MIPS and memory varying from 1600K to 4000K. A reduction in number of utility background jobs had little effect on increasing the throughput of the system. By decreasing the number of realtime applications, there was an appreciable increase in system throughput.

##### Results

Reduction of utility background jobs resulted in a net reduction of 1.6% of the time required to process the STIL job load, while a reduction of realtime jobs netted a reduction of 8% in STIL utilization. Reduction of realtime simulation requirements can, therefore, cause significant changes in STIL hardware requirements.

#### 5.4.3.4 Model Result Summary

From the two charts shown in Figure 5-22, an optimal CPU/memory size combination was derived. This combination was based on actual memory used to perform the required task and the percentage of jobs having to wait in a queue for execution. The CPU/memory combination from the charts appears to be 3 MIPS CPU power and 2500K memory size. But to effectively bound a CPU/memory combination, some room for growth must be considered. For this modeling study, a growth factor of 60% was assumed; therefore, from Figure 5-21, the CPU/memory combination that effectively handles the job load and provides the 60% growth potential is a 3 MIPS CPU power and a 3 million byte memory capacity.

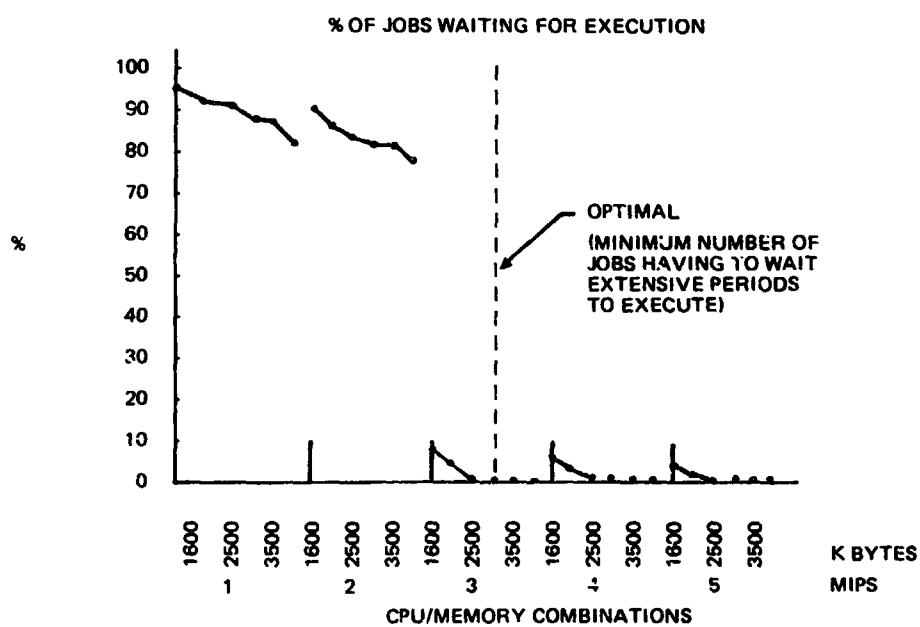
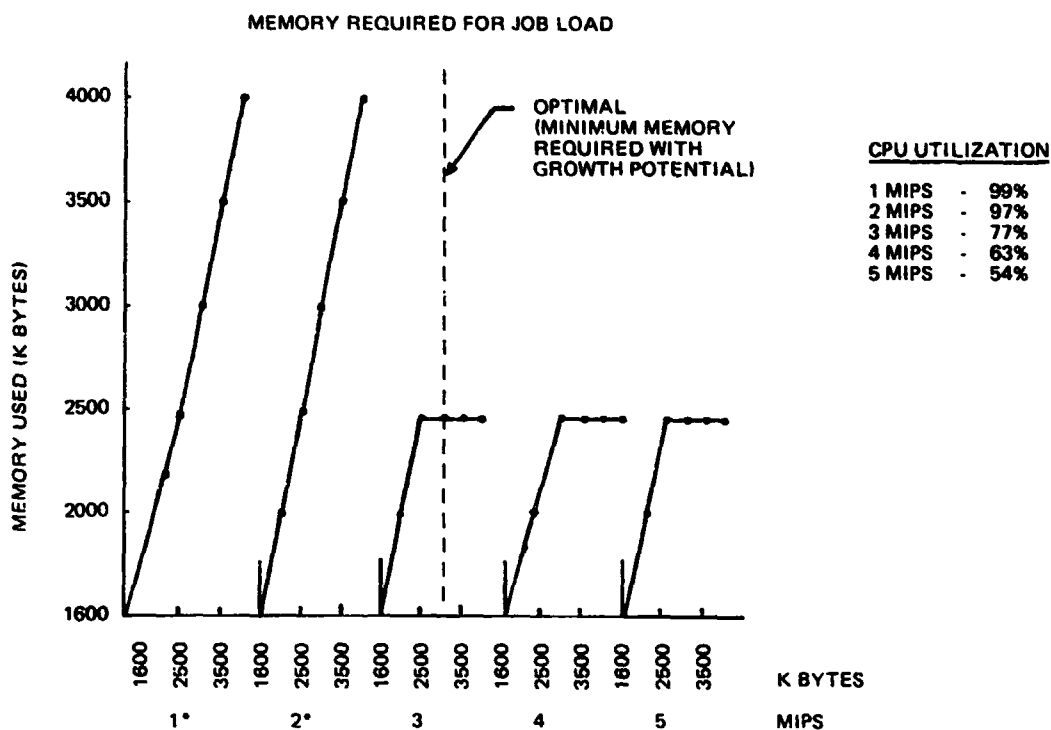


Figure 5-23. Selection of Optimal CPU Memory Size Combinations

## 5.5 STIL CONFIGURATION ANALYSIS

### 5.5.1 THEME

The design of a STIL hardware configuration must be customized to meet the requirements of the Spacelab program. Spacelab software development differs from that of previous space programs in the frequency of missions and variety of software-supported payloads. These and other factors will require a software development facility which will be able to process a large workload in a timely and cost effective manner.

The STIL modeling analysis provided the preliminary CPU and memory requirements which the STIL must support. This section will apply the modeling results with the projected workload estimates determined in other tasks to establish preliminary functional designs of potential STIL configurations.

### 5.5.2 CONCLUSIONS

Based on the analysis performed during this task, the following conclusions have been reached:

- A representative STIL configuration will have the following characteristics:
  - Processor CPU capability of 2.6 - 3 MIPS
  - Processor memory of 2.5 - 3 million bytes
  - 6 disk drives/8-10 magnetic tape drives
  - 24 remote terminals
  - 1-2 realtime interactive terminals
  - 4-5 printers
  - 2 card reader/punch
  - CID to support the host/CDMS interface
  - A Spacelab CDMS
- Many potential configurations exist--each with its own advantages/disadvantages
- Future detailed configuration studies should be conducted to apply upgraded STIL requirements to establish the most optimum configuration from cost, growth potential, and capability standpoints.

### 5.5.3 DISCUSSION

The selection of a STIL configuration must include consideration of Spacelab program requirements, many of which are still in a state of change. The approach of the study has been to develop estimates of the STIL workload and bound the requirements of the more critical STIL elements. The workload estimates have been based on currently available information

including the Shuttle traffic model as well as previous experience in aerospace software development. The critical STIL configuration elements were determined to be central processing (CPU), computing power, and memory size. Peripheral devices such as disks, tapes, and printers are less important elements due to the relative ease with which they can be added to or removed from a STIL configuration.

Within any STIL configuration, CPU computational capability must be considered the most critical element. A configuration design must provide adequate growth potential to ensure utilization of the host computer complex throughout the Spacelab program without a CPU change. To provide for potential growth, two design options are available. The first option is to design a configuration having multiple CPUs with capability to add or substitute larger, more capable CPUs. The second option is to design a configuration with a large single CPU with the capability to later substitute a larger CPU.

#### 5.5.3.1 Functional Configuration Analysis

Based on the STIL modeling results and the STIL requirements established in Tasks 2B, 3B, and 5 of the study, a functional STIL configuration was defined. The representative functional configuration for STIL is shown in Figure 5-24. As can be seen in the figure, the representative configuration provides the following capabilities:

- Processor with 3 MIPS CPU and 2.5 million bytes of memory (6 disk units)
- Extensive remote terminal support capability (up to 24 terminals)
- Realtime interaction support capability (1-2 terminals)
- Input/output capability of 8-10 magnetic tapes, 4-5 printers, and 2 card reader/punches
- A CID for host/CDMS interface

The CID provides the capability to perform realtime simulations utilizing the flight software sets within the CDMS computers. Because of the CID's criticality to the STIL's support capability, it will be discussed in more detail in the following paragraphs.

#### Computer Interface Device (CID)

The CID provides all the interface logic required to connect the host computer to the CDMS data bus. Through this interface, the host computer software will provide to the CDMS software all inputs and outputs as they would appear in actual flight.

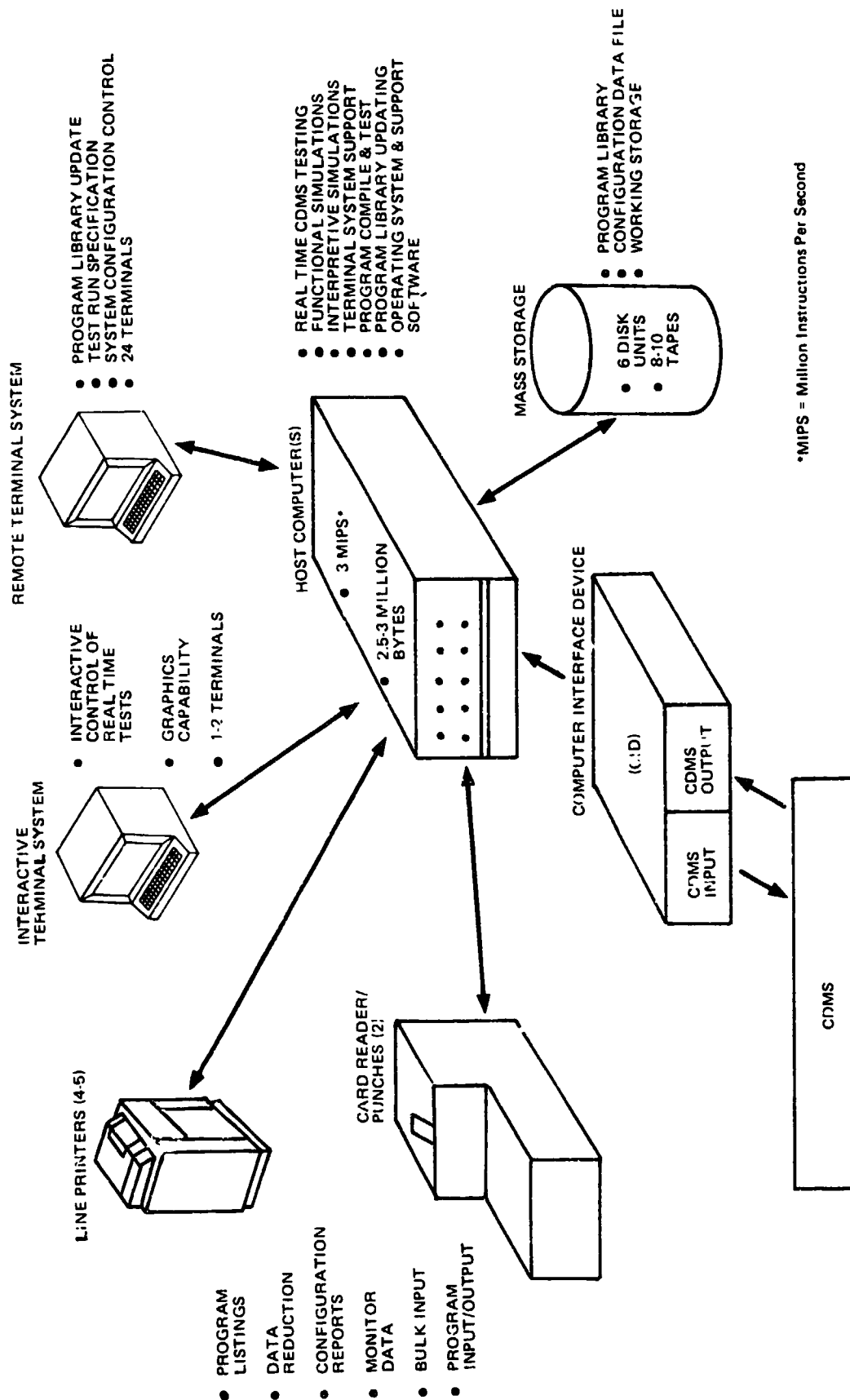


Figure 5-24. STIL Functional Diagram

In order to perform this function, the CID must provide the following operational interfaces:

- RAUs
  - Analog
  - Digital (discrete)
- Orbiter
  - Telemetry
  - Telecommand
  - PSS (depending on availability of an actual PSS station)
- EGSE
- CDMS computer controls

These interfaces are shown in more detail in Figure 5-25, and are discussed in the following paragraphs.

The RAU simulation will be accomplished by connecting the data bus to the CID and having the CID respond to data bus addressing and data transfers as would an RAU. For CDMS RAU input, the host computer software must furnish data computed by the subsystem and experiment models to the CID. The CID must put this data on the bus in response to addressing and function codes generated by the bus controllers. For CDMS RAU output, the CID must accept data from the data bus according to address and function codes and store the data for reference by the host computer software. The host computer software must then route the data to the appropriate models and test monitoring functions.

The Telemetry Interface will provide a data buffering capability for each of the CDMS computers. Logic within the CID must signal the host computer software for periodic transfer of data from the buffer to host computer memory for host software processing.

The Telecommand Interface must provide a capability whereby the host computer can issue simulated uplink commands to the subsystem and experiment computers through their respective bus controllers. These commands will be generated by host software at the specification and direction of the test conductor.

The PSS Interface is optional depending on the requirements to simulate the Shuttle Orbiter PSS. A simulation would be possible by utilizing a general purpose CRT console attached to the host computer. Display data addressed to and from the PSS station could be routed through the CID with data conversions performed as required by the host computer.

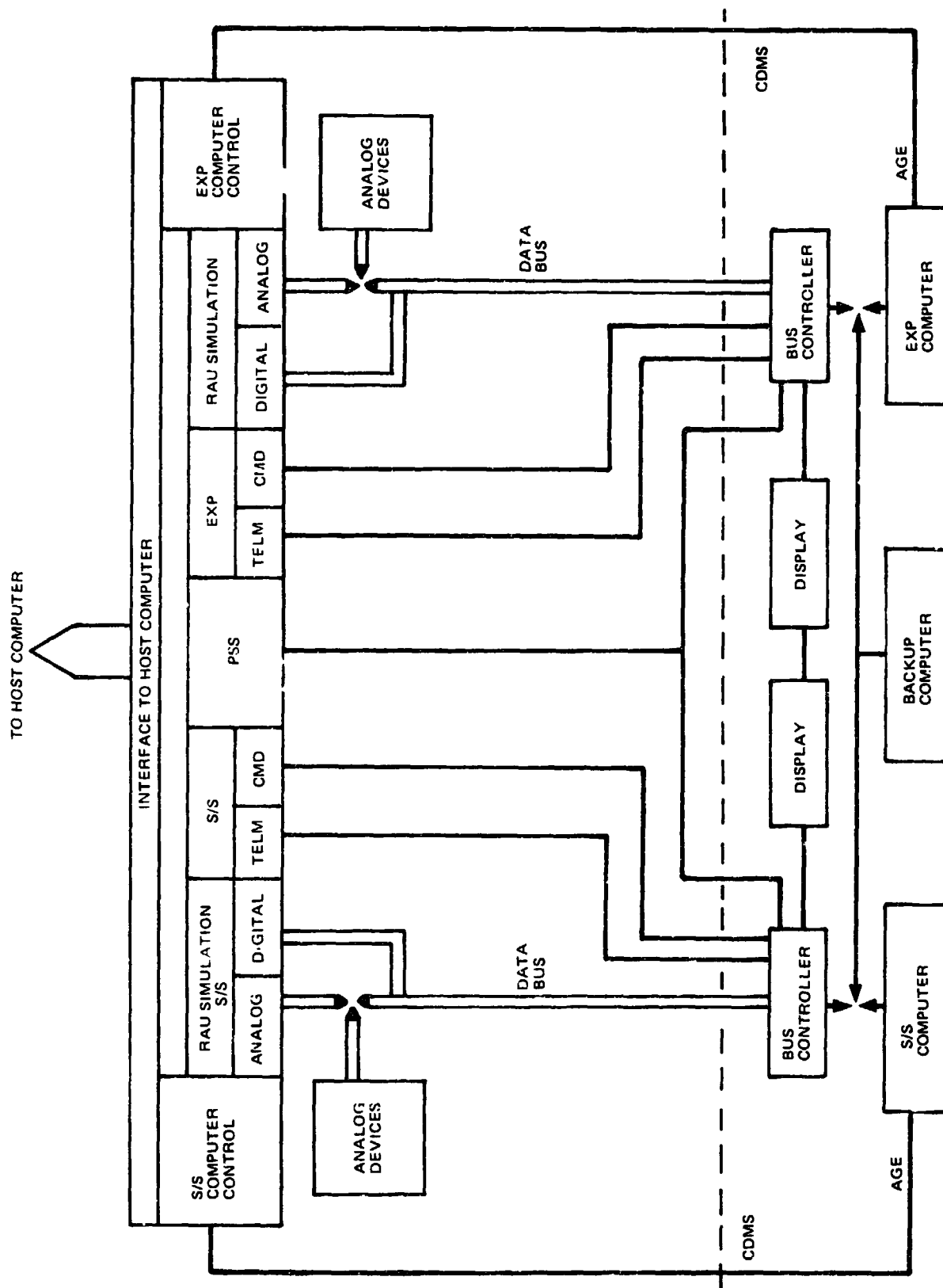


Figure 5-25. Computer Interface Device (CID)/CDMS Interface (Preliminary)



The RAU Analog Input simulation will provide the capability to attach actual analog hardware to the CID during a simulation. These analog devices would replace the digital models provided by the host computer software and would allow limited hardware/software integration testing within the STIL.

The EGSE interface will be simulated to provide the capability to fully integrate EGSE and CDMS software sets.

The CDMS Computer Controls must be manipulated by the host computer via the CID so that the CDMS computers can be initialized and controlled. This control must be effected primarily through the Aerospace Ground Equipment (AGE) control lines of the CDMS computers. Through use of these control lines, the host software can perform the following functions:

- Load and verify subsystem and experiment computer memories.
- Control the computers for start/sto, .
- Monitor computer registers for trace and compose stops.
- Dump computer memories for restart and print.

#### 5.5.3.2 Candidate STIL Configuration Concepts

Four candidate STIL configuration concepts have been established during this task. They are shown in Figures 5-26 through 5-29, and each is briefly discussed in the following paragraphs.

The configuration illustrated by Figure 5-26 depicts a centralized processing capability within a single large scale computer. Both the realtime and batch processing modes would be supported by this single processor. This configuration has the advantage of no inter-CPU communication or complex divisions of CPU responsibilities; however, growth capability will be limited to the capabilities of the computer selected.

As the Spacelab program matures, the contention for resources on the STIL may increase to the point that the single processor may become marginal in its ability to support requirements. This configuration would not be adaptable to such an environment, and procurement of a larger single processor would be required to support the processing load. This could severely impact schedules during the operational phase of Spacelab.

The configuration shown in Figure 5-27 features two processors with one processor dedicated to support of realtime simulation and the other dedicated primarily to batch and supportive functions. Communications between the processors will be via channel-to-channel and/or common disk storage. Such a configuration will result in use of smaller processors because of the load sharing. An additional feature of this configuration

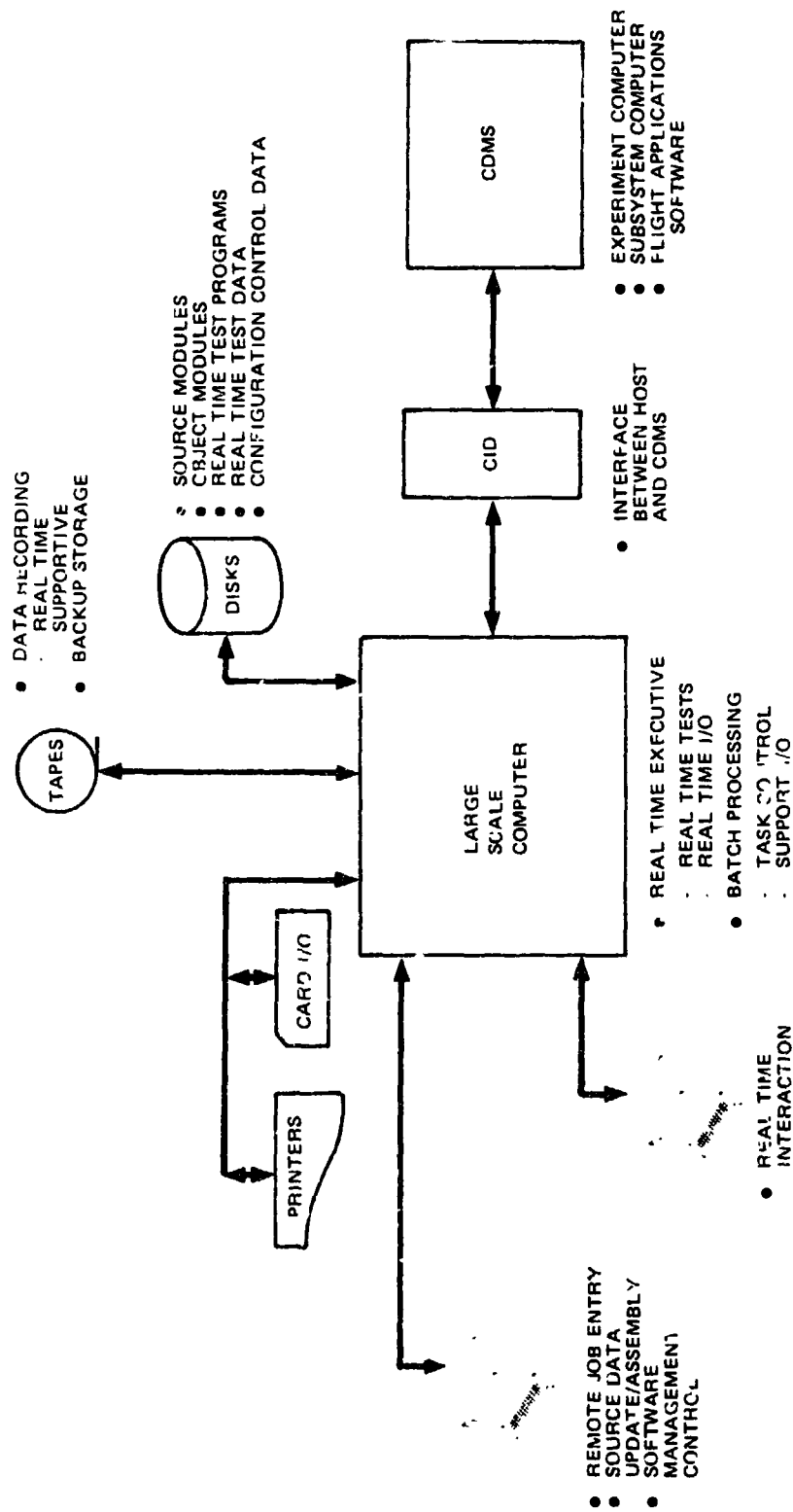


Figure 5-26. Centralized Single CPU Configuration

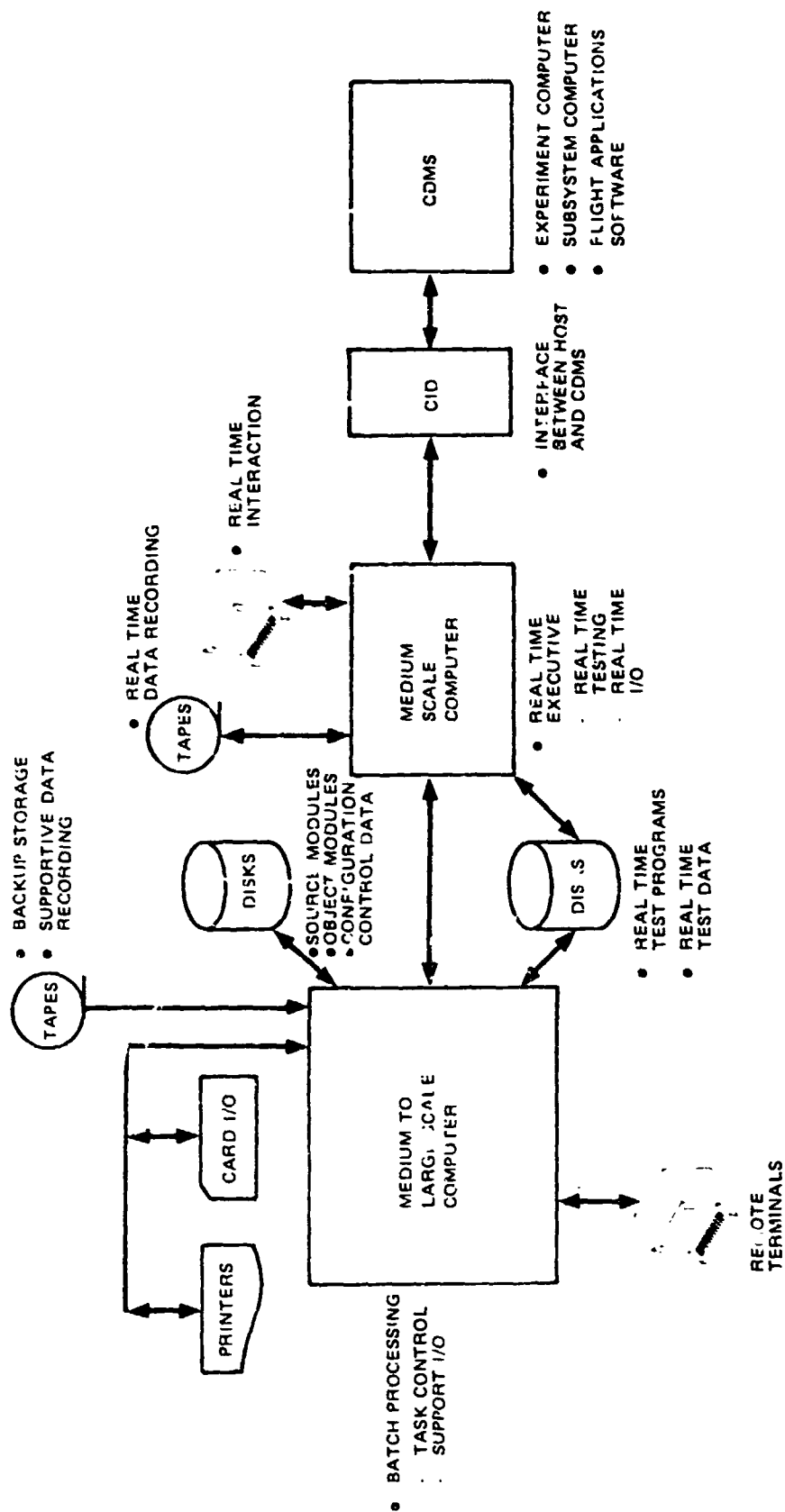


Figure 5-27. Decentralized CPU Configuration with Dedicated Real Time Processor

is the possibility of utilizing existing NASA processors to build the STIL. The principal disadvantage is that this configuration is tailored specifically to the Spacelab program and has little flexibility for adaptation to other future potential users.

The configuration shown in Figure 5-28 features a front-end processor and a main processor. The front-end processor performs scheduling of tasks for the main processor and controls STIL input/output functions. The main processor performs the batch and realtime processing functions. Such a configuration will allow a readily expandable computation base through the addition of processors (shown in Figure 5-29). This flexibility has considerable merit in that it provides the capability to consolidate computing facilities for such functions as mission planning and crew training into one facility. The principal disadvantage of the configurations is increased complexity of the software within the front-end processor.

Although the configuration concepts discussed above have been established, many additional combinations exist which can satisfy the STIL requirements. Prior to selection of the actual STIL configuration, a detailed analysis should be performed to determine the configuration best suited from growth, cost, available resources, and utilization considerations.

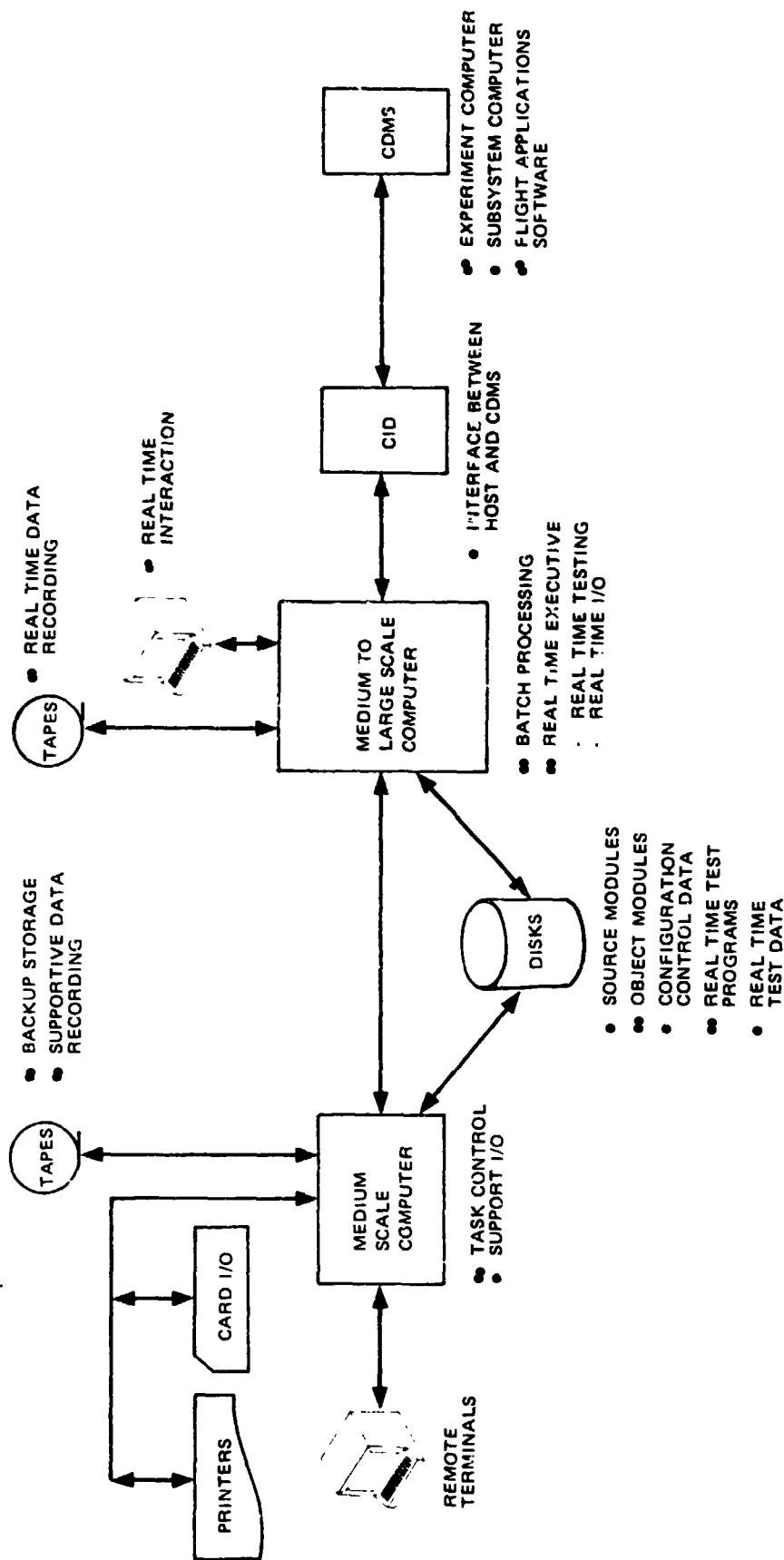


Figure 5-28. Decentralized CPU Configuration with Dedicated I/O Processor

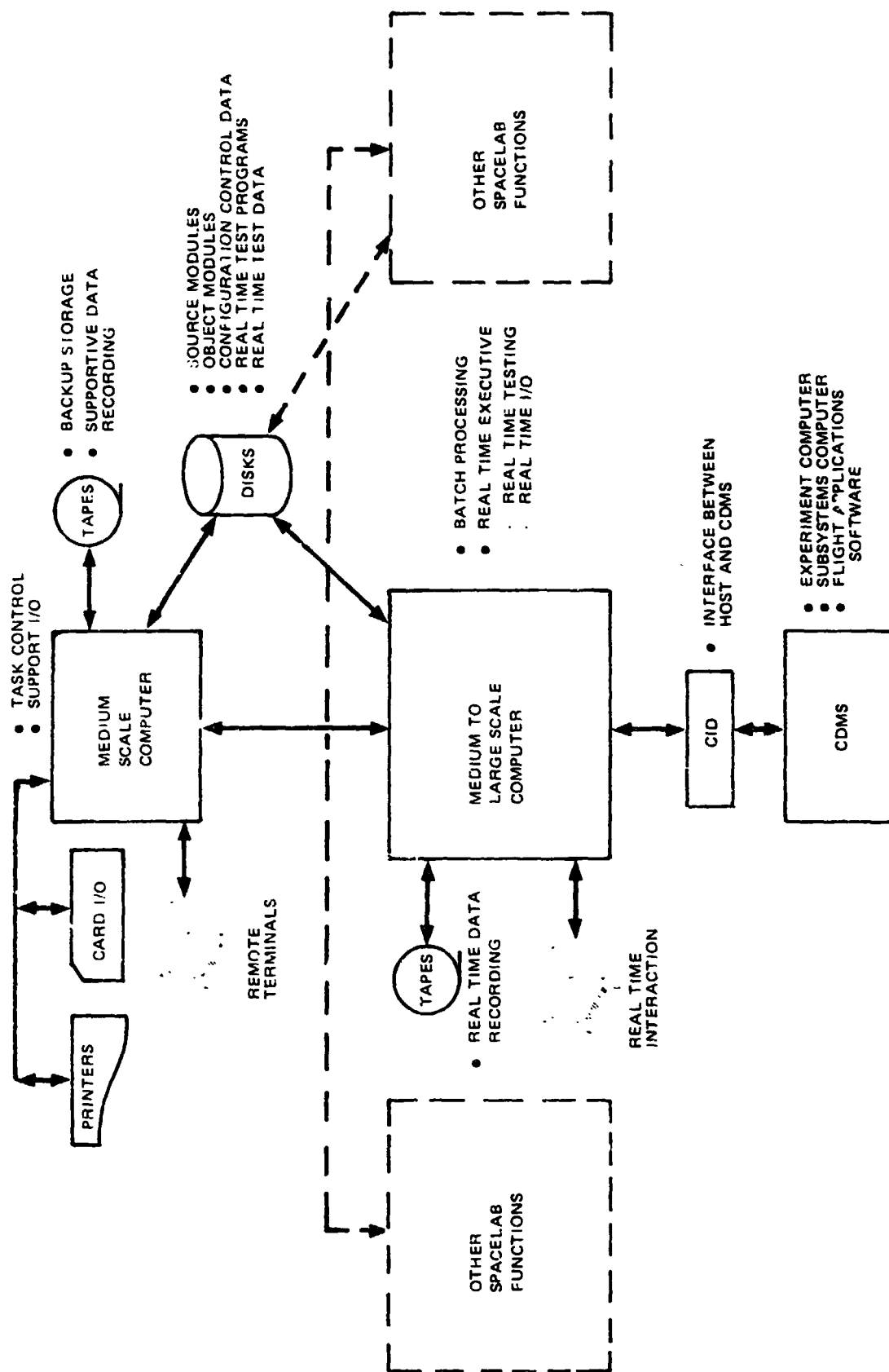


Figure 5-20. Decentralized CPU Configuration with Expansion for Other Spacelab Functions

## 5.6 STIL DEVELOPMENT PLAN ANALYSIS

### 5.6.1 THEME

To provide an orderly, systematic approach to STIL development, a development plan is an essential element. Within this study task, a preliminary development plan, based on projected overall Spacelab milestones, has been established.

### 5.6.2 CONCLUSIONS

Based on the development plan analysis conducted within this task, the STIL development plan must have the following characteristics:

- A definition task must be conducted to select the optimum configuration of STIL hardware to satisfy the overall STIL requirements.
- STIL software development for batch and supportive processing modes should begin upon selection of the STIL host computer.
- NASA will require computer services, with features compatible with selected host computers, for use in software development prior to receipt of host computer hardware.
- Operational support of realtime mode will be contingent upon the CID development plan, but batch and supportive modes must be available to support engineering model software integration and testing.

### 5.6.3 DISCUSSION

For establishing a STIL development plan, an analysis of the overall Spacelab Ground Operations Plan (Item 1, List of References) was conducted to establish the "need dates" for the STIL. Based on this analysis, the STIL development activities were divided into two major phases. The initial development phase (Phase I) included the definition and development of software to support the batch and supportive STIL requirements. The second phase (Phase II) included the development of the realtime simulation capability. The overall plan was keyed to STIL hardware milestones and to the need date established by delivery of the Spacelab engineering model (EM) and support software in 1978. The proposed overall STIL development plan is shown in Figure 5-30.

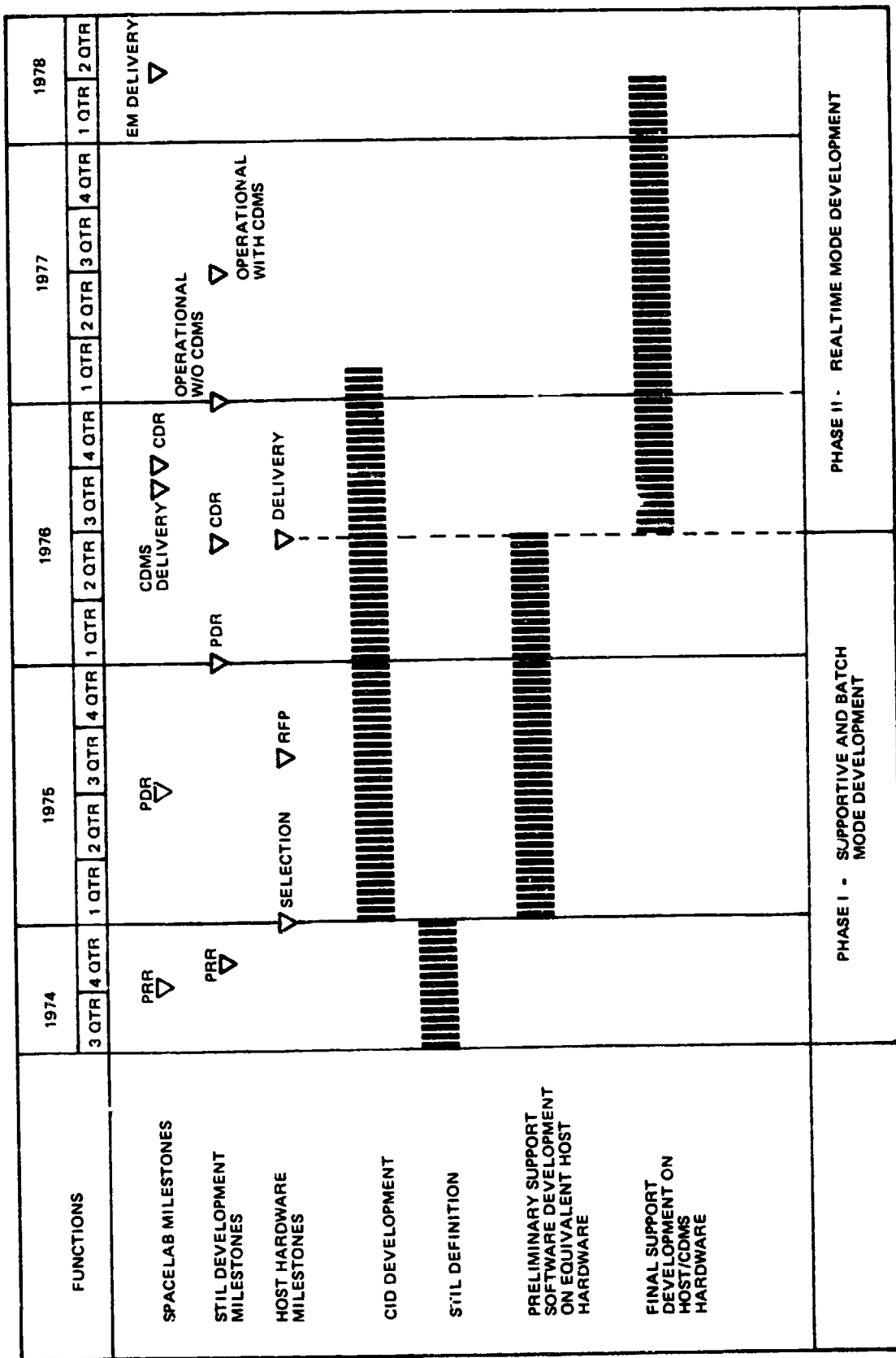


Figure 5-30. Preliminary STIL Software/Hardware Development Plan

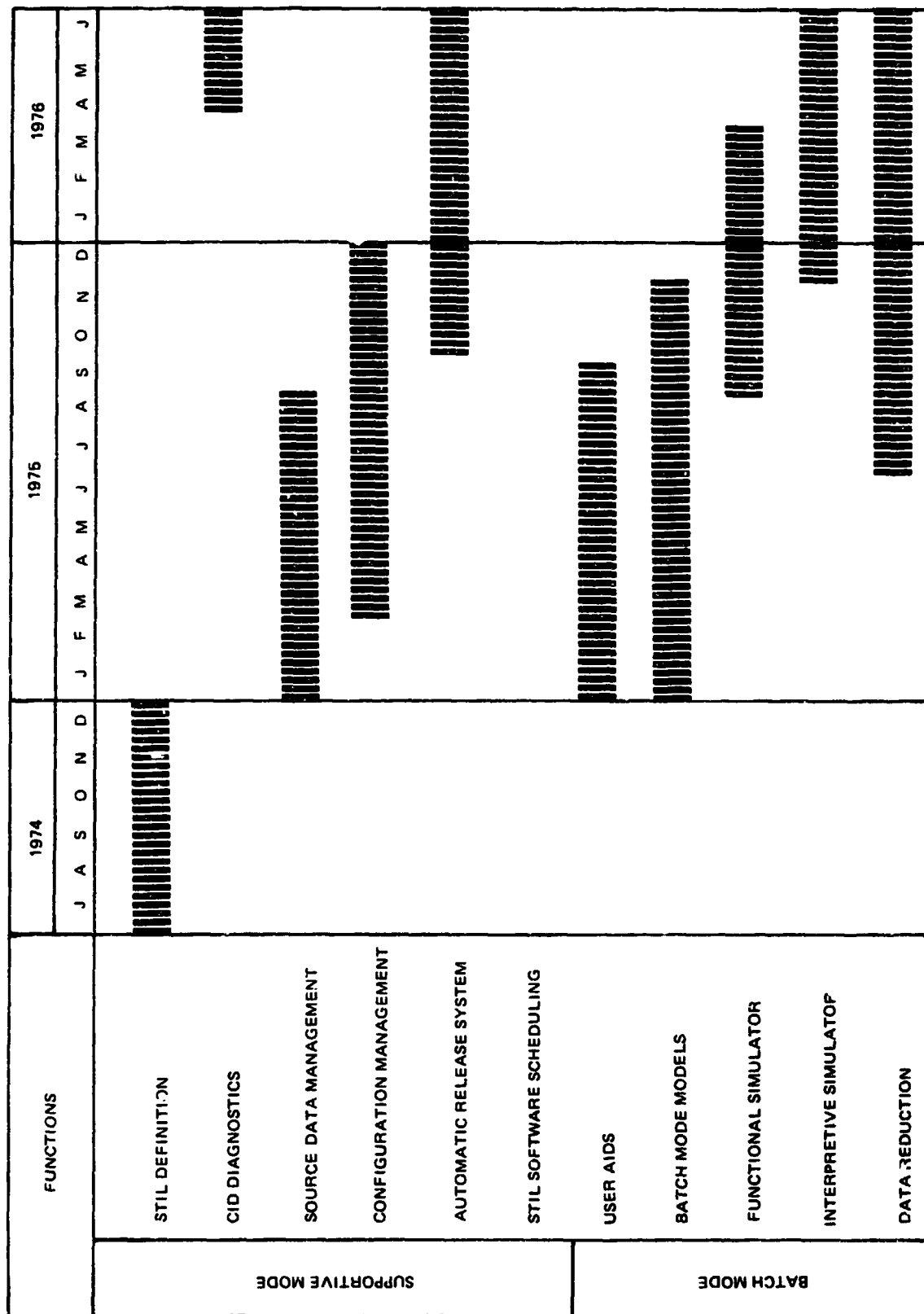


#### 5.6.3.1 Phase I Development

Phase I development encompasses the definition and computer configuration selection activity in which the actual host computer hardware is determined. Upon selection of the hardware, the development of the STIL software will be initiated. The software included in this phase will be that used to support batch and supportive processing modes. The Phase I development plan is shown in Figure 5-31.

#### 5.6.3.2 Phase II Development

Phase II development will be initiated upon receipt of the host computer and will be devoted largely to development of the realtime simulation capability. The Phase II development will terminate upon delivery of the engineering model in 1978. At this point, the STIL will be considered operational. The Phase II development plan is shown in Figure 5-32.



**Figure 5-31. Phase 1 - Preliminary Supportive and Datch Mobile Development Plan**



**APPENDIX A**

**SPACELAB SOFTWARE DEVELOPMENT  
AND INTEGRATION PLAN**

# Spacelab Software Development and Integration Plan

Prepared for the  
GEORGE C. MARSHALL  
SPACE FLIGHT CENTER  
Huntsville, Alabama

13 September 1974

Contract No: NAS8-30538

IBM No: 74W-00223

## Spacelab Software Development and Integration Plan

Classification and Content Approval

Robert C. Draney

Data Manager Approval

A. Radzowski

Program Office Approval

Fred D. Holdens

## PREFACE

This document defines the software development concepts and the management techniques to be used in the development and integration of Spacelab software. It describes the life cycle flow of the various components of Spacelab software, the management plan to control that flow, and tools required to support that flow. The scope of the plan includes the integration of ESRO developed software onto the NASA Software Test and Integration Laboratory and the development of experiment flight application software by and for the PI.

The plan was developed by the IBM Federal Systems Division, Huntsville, Alabama under contract no. NAS8-30538 from the National Aeronautics and Space Administration, Marshall Space Flight Center.

## TABLE OF CONTENTS

1.	INTRODUCTION . . . . .	1-1
1.1	Scope of the Plan . . . . .	1-1
1.2	Plan Structure . . . . .	1-1
1.3	Spacelab Software Terminology . . . . .	1-2
1.4	Reference Documentation . . . . .	1-2
2.	SPACELAB SOFTWARE MANAGEMENT CONCEPTS . . . . .	2-1
2.1	Standardized Modular Software Development . . . . .	2-1
2.2	Software Management Control . . . . .	2-1
2.3	NASA Integration of Existing Software . . . . .	2-2
2.4	Experiment Flight Application Software Development . . . . .	2-2
2.5	NASA Provided Software Test and Integration Laboratory (STIL) . . . . .	2-4
3.	SPACELAB SOFTWARE LIFE CYCLE FLOW . . . . .	3-1
3.1	Basic Software Development Life Cycle Flow . . . . .	3-1
3.1.1	Definition Phase . . . . .	3-1
3.1.2	Development Phase . . . . .	3-1
3.1.3	Operational Phase . . . . .	3-3
3.2	Spacelab Software Schedule . . . . .	3-3
3.2.1	Software Test and Integration Laboratory (STIL) Software Schedule . . . . .	3-3
3.2.2	Payload Operations Center Software Schedule . . . . .	3-3
3.2.3	Preprocessing Facility Software Schedule . . . . .	3-3
3.2.4	Experiment Applications Software Schedules . . . . .	3-5
3.3	STIL and ESRO Software Integration . . . . .	3-5
3.3.1	STIL Activation . . . . .	3-5
3.3.2	Spacelab Software Training . . . . .	3-7
3.3.3	EGSE Support Software Installation . . . . .	3-7
3.3.4	EGSE Ground Checkout Installation . . . . .	3-7
3.3.5	CDMS Support Software Installation . . . . .	3-7
3.3.6	EM Software Installation . . . . .	3-8
3.4	Operational Software Development Life Cycle Flow . . . . .	3-8
3.4.1	Definition Phase . . . . .	3-8
3.4.2	Development Phase . . . . .	3-8
3.4.3	Operations Phase . . . . .	3-10



## TABLE OF CONTENTS (CONTINUED)

		<u>Page</u>
4	EXPERIMENT FLIGHT APPLICATION SOFTWARE DEVELOPMENT PLAN. . . .	.4-1
4.1	Experiment Application Software Control . . . . .	.4-1
4.2	Experiment Application Software Responsibilities and Integrations. . . . .	.4-3
5	SOFTWARE DEVELOPMENT STANDARDS AND TECHNIQUES. . . . .	.5-1
5.1	Software Definition and Design Phase. . . . .	.5-1
5.2	Software Implementation Phase . . . . .	.5-1
5.3	Verification/Validation Phase . . . . .	.5-2
5.4	Documentation Standards . . . . .	.5-2
6	SOFTWARE CONFIGURATION MANAGEMENT. . . . .	.6-1
6.1	Configuration Control . . . . .	.6-1
6.1.1	Spacelab Software Configuration Control Board (CCB) . . .	.6-1
6.1.2	Spacelab Software Processing Control Flow . . . . .	.6-1
6.2	Configuration Identification. . . . .	.6-2
6.3	Configuration Accounting. . . . .	.6-2
7	SOFTWARE DEVELOPMENT SUPPORT TOOLS . . . . .	.7-1
7.1	STIL Support Software Requirements. . . . .	.7-1
7.2	STIL Hardware Requirements . . . . .	.7-3

This document establishes a baseline Spacelab Software Development and Integration Plan. The plan provides a basis for NASA to use in planning for the development of Spacelab Application Software and for integrating the software developed by PIs and ESRO into the Spacelab operational environment. This plan defines concepts and techniques for providing software which will be easily integrated and maintained by NASA.

### 1.1 SCOPE OF THE PLAN

This plan is intended to cover all Spacelab operational software but is primarily directed toward computer software which will execute on the Spacelab CDMS computers, EGSE computer, and STIL host computer.

### 1.2 PLAN STRUCTURE

The plan is structured to give the reader a quick understanding of the basic concepts recommended for Spacelab software development and integration. Concept and technique details and rationale can be found in the body of this report and in the reference documentation. The following paragraphs summarize remaining sections of the plan.

Section 2 - Spacelab Software Management Concepts documents the concept of NASA assuming maintenance responsibility and operational support of ESRO-developed software and for minimizing development costs of NASA-developed software.

Section 3 - Spacelab Software Life Cycle Flow provides a description of the basic software development cycle, gives the Spacelab software schedule, discusses the initial integration of ESRO software on the STIL, and describes the operational software development cycle.

Section 4 - Experiment Flight Application Software Development Plan describes the management of the options provided the PI for the development of experiment support software. Management of these options is highlighted because development of the Experiment Flight Applications software will be a continuing process during the Spacelab life cycle.

Section 5 - Software Development Standards and Techniques lists those state-of-the-art development standards and techniques recommended for usage by ESRO as well as NASA during software development.

Section 6 - Software Configuration Management describes the management concepts which are as important in software as in hardware development and operation. The techniques presented in this section will provide sufficient NASA control and visibility for Spacelab software.

Section 7 - Software Development Support Tools describes the tools that are important in manufacturing and maintaining software. This section lists the tools recommended for NASA utilization in the Spacelab program.

### 1.3 SPACELAB SOFTWARE TERMINOLOGY

Software terminology within large scale software developments is similar and usually varies only in the definition of overall system design and structure. Within the design of Spacelab, a basic hierarchical structure has been established which will provide a method of describing software interrelationships and interactions between the major CDMS and EGSE software.

For the purposes of this plan and future effective communications, the software hierarchy presented in Figure 1-1 will be used. The relationships (from the lowest identified element to the highest) can be summarized as follows:

Module is the lowest element of software to be under configuration control. The module is considered to contain approximately 100 HOL program statements and is a basic Spacelab building block. The module can be considered analogous to a circuit board or chip in hardware.

Package is a combination of modules into a logical unit to satisfy the requirements of a particular function. An example of a package is those modules which make up a flight application function. In hardware, this would be analogous to a major subassembly.

Set is a combination of packages to satisfy the requirements of a payload. An example of set would be the combination of flight applications and operating system packages to form the software to execute in the experiment CDMS computer while in orbit. In hardware, this would be referred to as a major subsystem. Sets can be combined to form sets of sets as in a hardware sense subsystems together form systems.

As can be seen in Figure 1-1, the Mission Software Set is composed of the CDMS Flight Set, the CDMS Ground Checkout Set, and the EGSE Ground Checkout Set. Each CDMS set, correspondingly, is composed of an Experiment CDMS and Subsystem CDMS Set. The figure thus establishes the logical relationships among modules, packages, sets and sets of sets.

### 1.4 REFERENCE DOCUMENTATION

- o IBM, Study on Spacelab Software Development and Integration Concepts Final Report, IBM No. 74W-00223, August 15, 1974.
- o NASA, Spacelab Ground Operations Plan, 68M00032, March 28, 1974.
- o IBM, Program Management Plan for the Spacelab Software Development and Integration, IBM No. 74W-C0105, April 4, 1974.
- o IBM, Spacelab Software Development and Integration Concepts Study Report - Volume I and II, IBM No. 73W-00326, October 31, 1973.
- o NASA-ESRO, Spacelab Programme Requirements Level I, March 5, 1974.

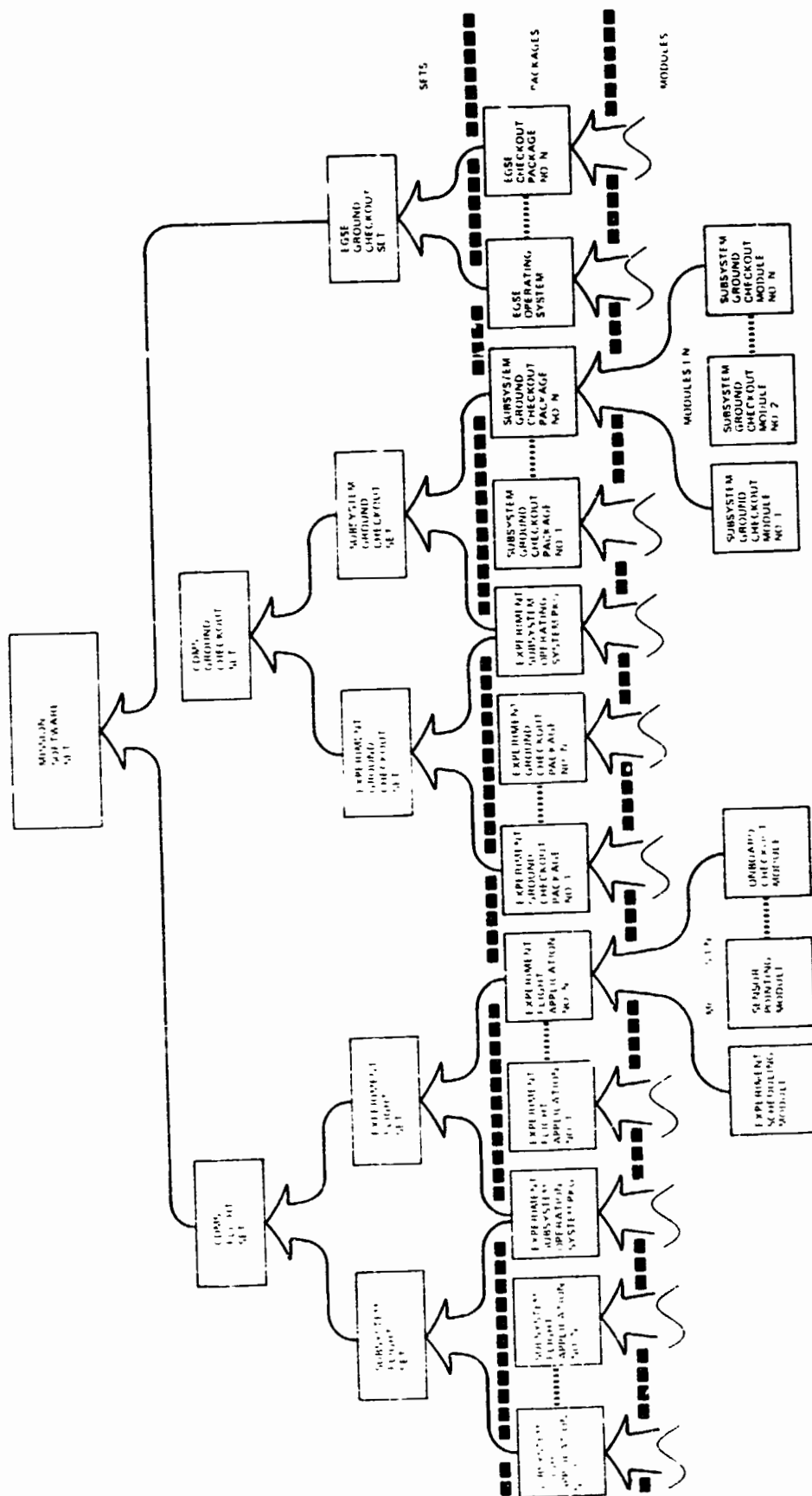


Figure 1-1. Example of Spacelab Software Hierarchical Structure

- o NASA-ESRO, Spacelab System Requirements Level II, March 1, 1974.
- o IBM, Spacelab Data Management Study, IBM No. 73W-00314, October 15, 1973.
- o IBM, Spacelab Sortie Payload Sizing Analysis, Contract No. NAS8-14000, IBM No. 74W-00059, DRL No. 1615, February 27, 1974.
- o NASA, Space Shuttle Program Level II Program Definition and Requirements Volume XVIII, NASA-JSC-07700, Volumes IV, IX, and XVIII.
- o ERNO, Proposal For the Spacelab Design and Development Contract to ESRO/ESTEC RFP A0/600 Volume 1 - Technical Proposal, April 16, 1974.
- o IBM, Space Shuttle Orbiter Avionics Software Management Plan, IBM No. 73-SS-0021, October 22, 1973.

## SPACELAB SOFTWARE MANAGEMENT CONCEPTS 2

Software management, like hardware management, results in a high quality product on schedule and within cost. Software is controllable and can be managed in much the same manner as hardware. Spacelab software management is characterized by its massive size and by multiple organization involvement. Lacking strong and positive management, loss of control will occur and software integration will not be possible. Because Spacelab software integration will occur late in the Spacelab program, problems encountered in the integration process will result in schedule slips and cost impacts.

Through the proper application of software management concepts, Spacelab software objectives can be met with minimum cost and within schedule. To achieve low cost software development, maximum utilization of existing facilities, existing components, and proven development techniques must be employed. The Spacelab software management concepts described herein are intended to provide the most cost effective approach to software development by standardizing software building blocks, providing proper level of management control, maximizing use of existing software, and providing efficient centralized software manufacturing services.

### 2.1 STANDARDIZED MODULAR SOFTWARE DEVELOPMENT

The modular software development concept is analogous to the development and baselining of standardized circuit boards and then the use of these off-the-shelf boards to develop major system configurations. Spacelab software will be approached in the same manner, in that, common software modules will be baselined and used in developing Spacelab software.

To achieve this objective, NASA will standardize facilities and require that software developers (NASA and ESRO) utilize the standard modular software development approach. This approach will provide software modules which may be used in a cost effective manner across the Spacelab program.

To meet the objective of using software across the Spacelab program, early establishment of software development standards and procedures is required. State-of-the-art standards for software design, implementation, verification, and documentation will be imposed on all Spacelab software developers, resulting in an inventory of well documented, standardized software modules which are transferable across the Spacelab program.

### 2.2 SOFTWARE MANAGEMENT CONTROL

The management concept for software control provides flexibility during the software development process. Software developers will be required to work within proven effective software development standards which will ensure

modularity and the most cost effective integration of the software into the Spacelab program. Once the software modules have been verified and certified, management control is required to assure continued system compatibility and the highest reliability.

### 2.3 NASA INTEGRATION OF EXISTING SOFTWARE

ESRO will make available as part of the Spacelab, at a minimum, the following major software building blocks:

- o Common CDMS Operating System Package
- o CDMS Ground Checkout Packages
- o Subsystem Flight Application Packages
- o EGSE Computer Operating System Packages
- o EGSE Ground Checkout Packages
- o Support Software Used in Development of Spacelab Software
- o Models and Simulation Systems Used in Verifying Software.

This software will be baselined and installed on NASA development facilities with little or no modifications and will be maintained by NASA throughout the software life cycle. NASA has developed a significant amount of related software for the Saturn, Apollo, Skylab, and Shuttle programs which will be utilized as much as possible when activating a Software Test and Integration Laboratory (STIL), Payload Operation Center (POC) and Preprocessing Facility (PPF). Existing software availability should be a prime consideration when selecting the facility hardware.

### 2.4 EXPERIMENT FLIGHT APPLICATION SOFTWARE DEVELOPMENT

The Experiment Flight Applications software, as well as the experiment hardware will be undergoing continuous development through the Spacelab life cycle. It is the intent of NASA to allow the Principal Investigator (PI) maximum flexibility in developing the required software to operationally support his experiment. Figure 2-1 illustrates the five development options available to the PI. Due to the fact that several PIs and experiments may be involved in a single mission, NASA will be responsible for integration and verifying the Experiment Flight Set on the STIL and then validating the set on the hardware at the Central Integration Site (CIS) prior to shipment to the launch site. It is NASA's intent that the PI be always fully responsible for any software required to test hardware prior to hardware integration at the CIS.

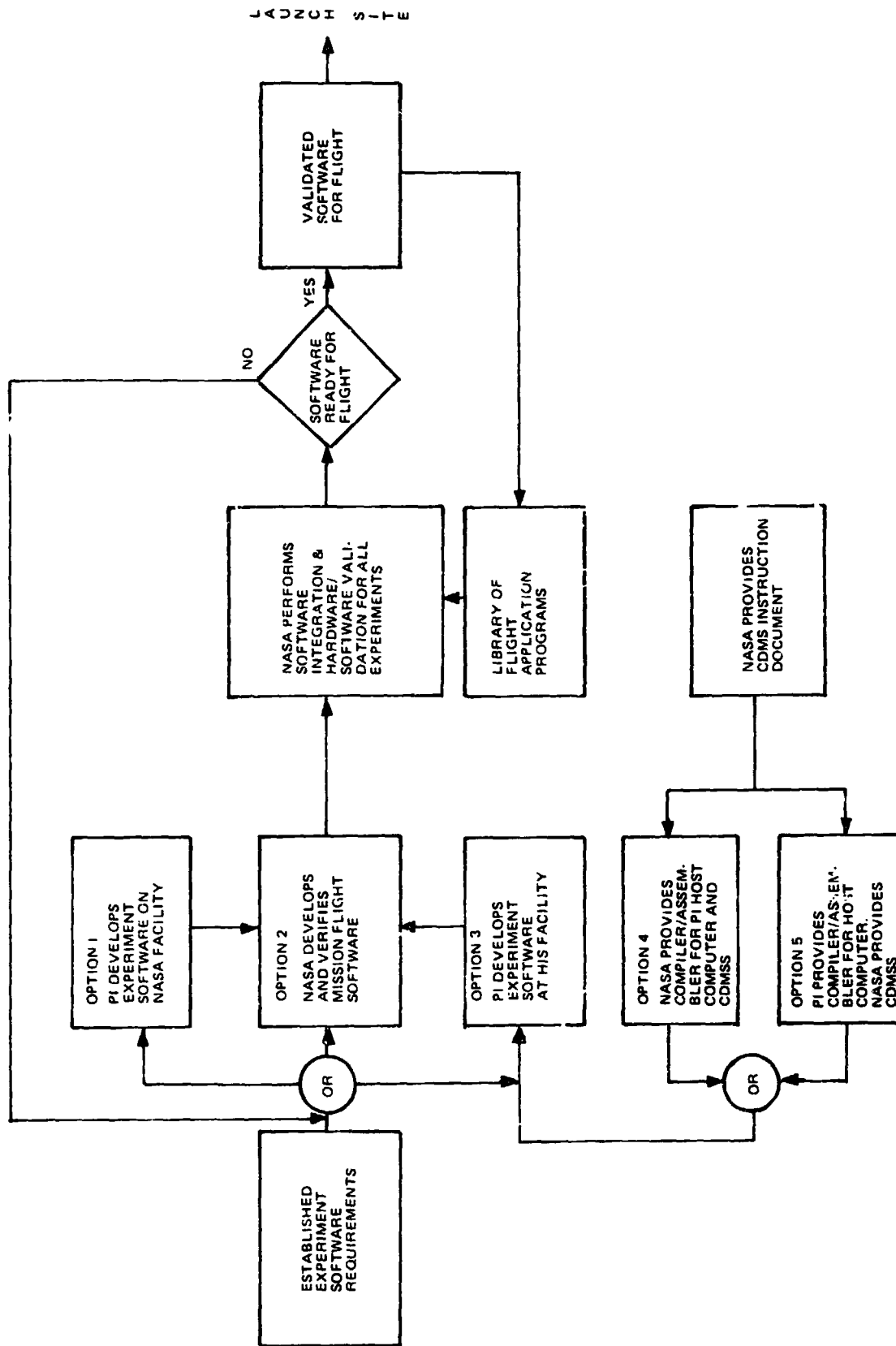


Figure 2-1. Experiment Software Development Options



## 2.5 NASA PROVIDED SOFTWARE TEST AND INTEGRATION LABORATORY (STIL)

To provide the environment for the development of low cost Spacelab software, NASA will provide a state-of-the-art software development and integration laboratory. This facility will be designed to fully support those standards, techniques, and procedures which have proven to be most cost effective in software development.

## SPACELAB SOFTWARE LIFE CYCLE FLOW 3

Spacelab software life cycle flow represents the various work efforts and interrelationships from the time that a software need is identified until that need is fully satisfied. In general, the Spacelab software life cycle is the same as that for any large software system. As in hardware system development, software system development has a top level flow and various intermediate level flows with all levels having the same basic functions. An example is that the Spacelab has a PRR, PDR, CDR and Acceptance, and each subsystem within the Spacelab has a similar cycle.

It is the intent of this section to describe the Spacelab software life cycle flow as a basic cycle and then relate this cycle at the level of major Spacelab software sets (i.e., Experiment Flight Application Set, Subsystem Flight Set, CDMS Ground Checkout Set, EGSE Ground Checkout Set, Payload Operations Center Sets, Preprocessing Facility Sets, and STIL Support Software Sets). The logical time flow of this software will be presented as well as its operational flow through the Spacelab facilities. The integration of the ESRO developed software into the Software Test and Integration Laboratory (STIL) is included within this section.

### 3.1 BASIC SOFTWARE DEVELOPMENT LIFE CYCLE FLOW

The basic software life cycle flow consists of the three major phases of definition, development and operation. As can be noted in Figure 3-1, these phases are broken up into several subphases. As in hardware development flow, the software flow activity results first in baseline requirements, then in design and finally in the end item. This basic flow can be used at the module, the package, or the set level of software manufacture.

#### 3.1.1 DEFINITION PHASE

The definition phase is initiated through user identification of requirements of the software system to be developed. At first this is done at a high system level, and functions and requirements are allocated to facilities hardware and software. The level of detail increases as requirements are converted into measurable and understandable terms such as performance criteria, display formats, logical sequences, and algorithms to be solved. The definition phase results in a firm requirements baseline at the PRR. This phase is normally the responsibility of the system designer or PI at the package or set levels or the software designer at the module level.

#### 3.1.2 DEVELOPMENT PHASE

The software development phase consists of the software design, code and test, verification, and validation of the software. During the development phase, checks and balances (Quality Control) are provided to ensure the software

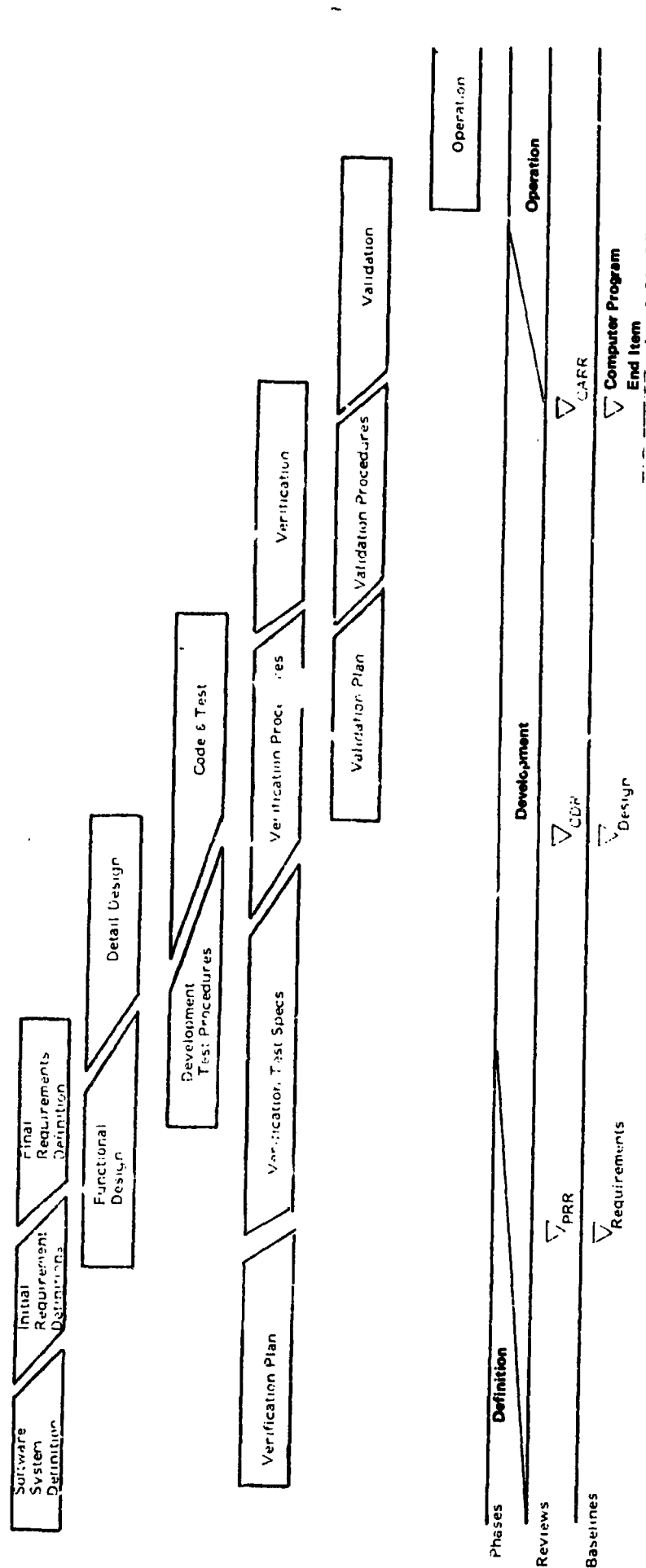


Figure 3-1. Basic Spacelab Software Life Cycle Flow

end item meets the needs of the user. These quality tests are called verification and validation. The verification assures that the software designed, coded and tested meets the specifications, and validation assures the hardware/software system meets the needs of the user. Emphasis on quality testing is varied depending on the level of integration (module, package or set). There are two major milestones in the development phase: CDR, where the design is baselined and CARR, where the software is put into operational status.

### 3.1.3 OPERATION PHASE

The software operation phase involves the use of the software following CARR. The operation may identify errors or shortcomings in the operational software which will require maintenance activities to be performed. This could result in a full life cycle or a portion of the life cycle depending on the magnitude of the required change.

## 3.2 SPACELAB SOFTWARE SCHEDULE

Overall software schedules and planning can be derived from Figure 3-2. As noted, major software definition activities are in process at the current time (August 1974). Many of these activities actually began during Phase B Spacelab studies. The operational milestones for NASA software are keyed to the EM delivery and first Spacelab flight.

### 3.2.1 SOFTWARE TEST AND INTEGRATION LABORATORY (STIL) SOFTWARE SCHEDULE

The STIL Operational date has been dictated by the delivery of the Spacelab Engineering Model (EM). The STIL must be operational in a time period to integrate the ESRO software into the STIL and to provide NASA support personnel with the training required to fully support ESRO developed software during EM hardware integration into the CIS. Additional detail on STIL activation can be found in Paragraph 3.3.

### 3.2.2 PAYLOAD OPERATIONS CENTER SOFTWARE SCHEDULE

The Payload Operations Center software must be operational early enough to train ground controllers prior to first flight of the Spacelab.

### 3.2.3 PREPROCESSING FACILITY SOFTWARE SCHEDULE

The Preprocessing Facility must be operational prior to the first Spacelab flight. As can be seen in the schedule, this development effort follows the other two initial efforts due to later operational date requirements.

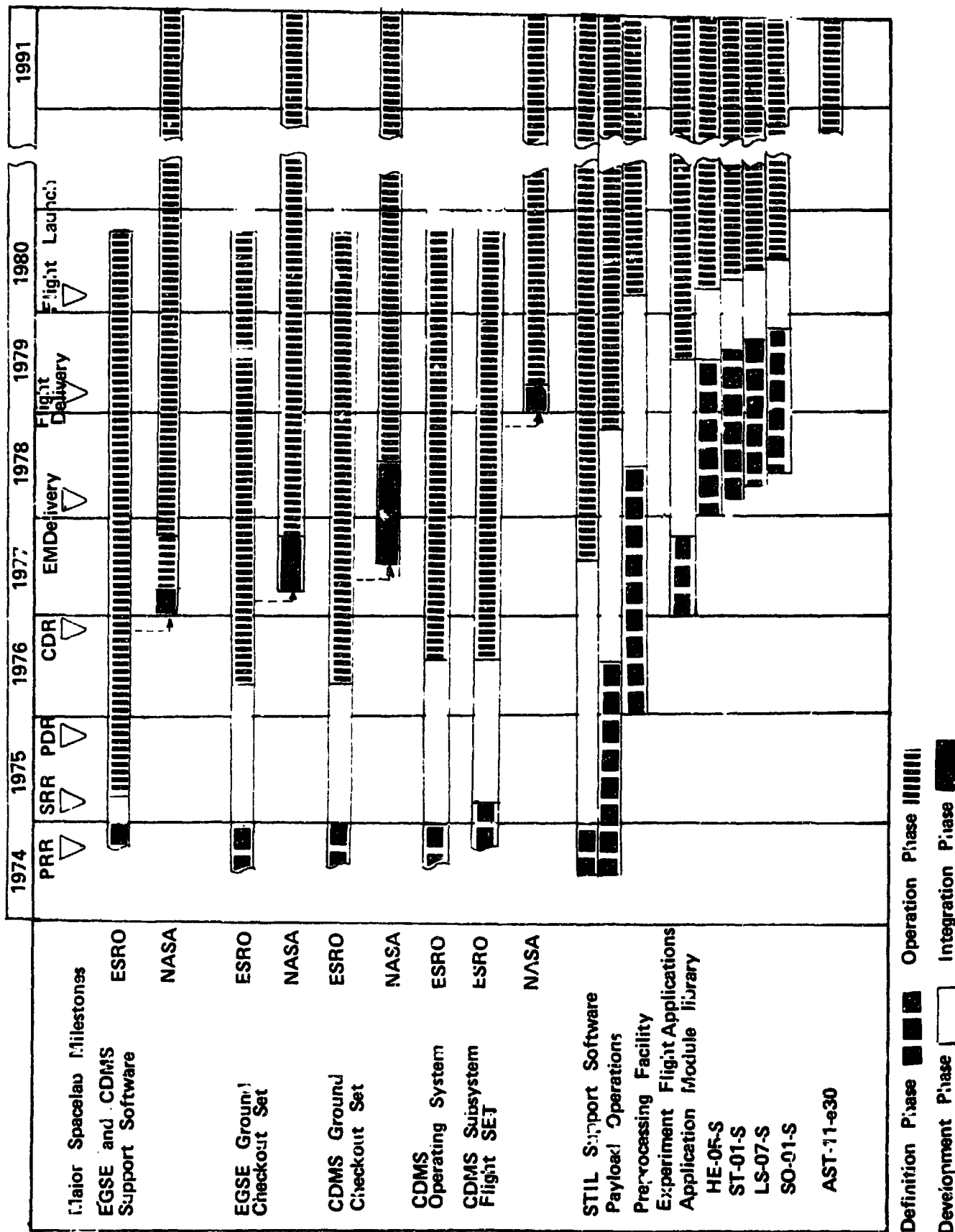


Figure 3-2. Spacelab Software Schedule

### 3.2.4 EXPERIMENT APPLICATIONS SOFTWARE SCHEDULES

Figure 3-2 indicates a relatively long definition phase for each experiment application followed by a six month development phase. This type life cycle will provide the capability to commit to software development phase after requirements are firm. The development cycle for common experiment application modules begins early to develop a library of building blocks.

### 3.3 STIL AND ESRO SOFTWARE INTEGRATION

It is imperative that NASA prepare for accepting the maintenance and operational responsibilities for Spacelab software. Facilities must be developed, personnel training must be completed, test procedures must be developed, and complete familiarity with the Spacelab software must be obtained prior to the EM being received by NASA.

This section of the plan addresses the basic steps in activating the Software Test and Integration Laboratory and preparing for acceptance of software maintenance responsibilities. The STIL is considered operational following the installation and verification of the Engineering Model software.

#### 3.3.1 STIL ACTIVATION

STIL activation encompasses all activities associated with the definition, development, installation, and verification of the facility. The key milestones in STIL activation are illustrated in Figure 3-3. A brief summary of activities leading to these milestones is contained in subsequent paragraphs. Details may be obtained from Section 5 of the Study on Spacelab Software Development and Integration Concepts Final Technical Report, NAS8-30538, IBM No. 74W-00224.

##### o Host Computer Availability

The first major milestone of STIL activation following the definition and detail design is the installation of the host computer complex. The complex includes the computer and memory, peripheral input/output devices, data base storage, graphics terminals, strip charts, plotters, and remote terminals. The STIL facility will then support software system activation and development.

##### o CDMS Delivery

Prior to the CDMS delivery from ESRO, the CDMS Interface Device (CID) must be defined, designed, manufactured, unit tested and interfaced with the host computer. Following CDMS delivery, the CDMS is integrated into the STIL, and development of the realtime interactive software continues utilizing the CDMS.

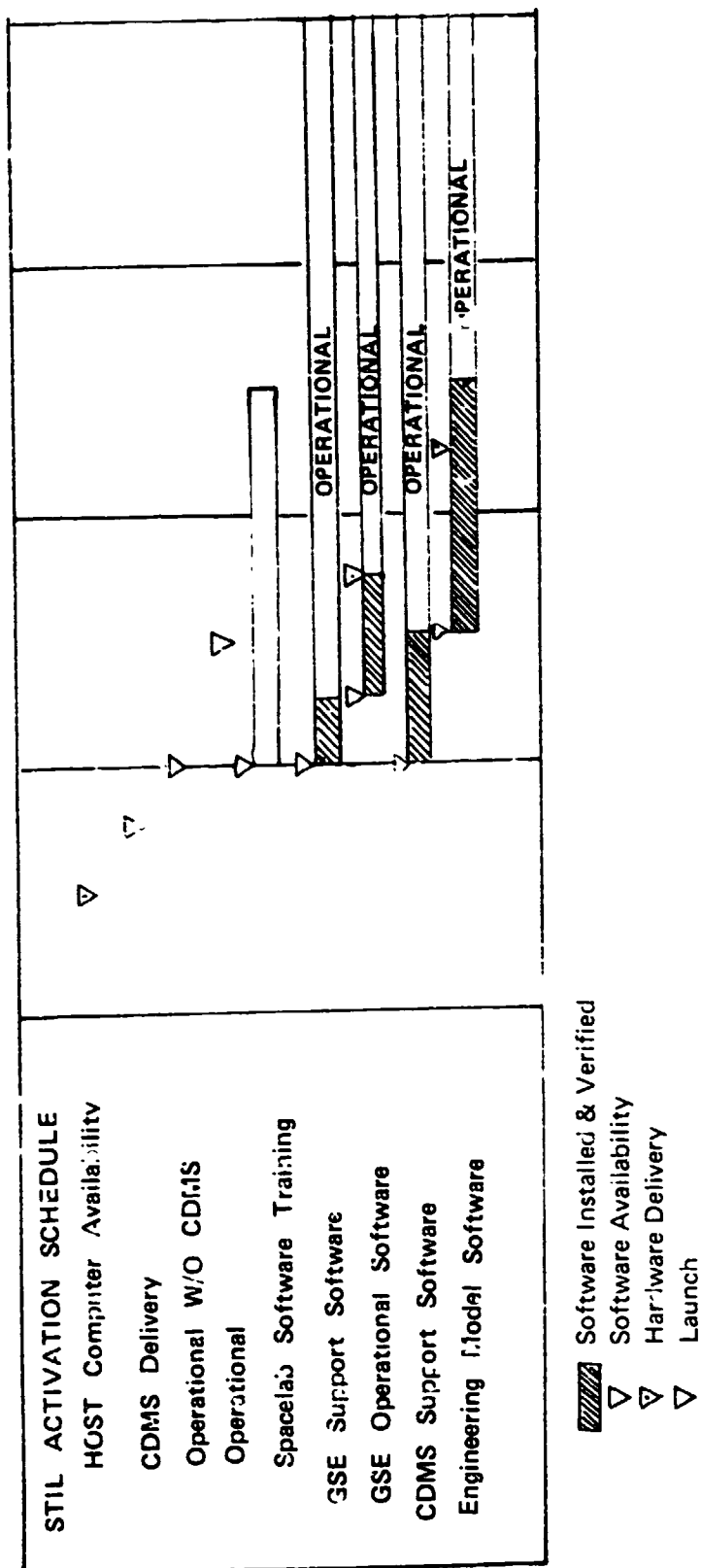


Figure 3-3. STIL/ESRO Software Integration Schedule

o STIL Operational Without the CDMS

Following the verification of the facility capability to support batch and supportive modes of operation, the STIL will be available for data base installation, personnel training, and the initial efforts associated with the receipt of ESRO software.

o STIL Operational

Following the development of the capability to support the CDMS in a realtime mode, the STIL is operational and is capable of supporting development and/or maintenance of Spacelab software.

### 3.3.2 SPACELAB SOFTWARE TRAINING

STIL operation training begins when the STIL first becomes operational without the CDMS. This includes the training for remote terminal operators as well as software designers and programmers. The terminal operators begin data base generation, STIL scheduling and configuration control procedures as soon as training is complete.

Additional training is provided to the software designers, PI's, programmers and verification personnel on the operation of the realtime mode following the STIL operational milestone.

### 3.3.3 EGSE SUPPORT SOFTWARE INSTALLATION

This activity consists of installing the ESRO supplied EGSE support software packages on the STIL and the training of computer operators. The EGSE support software includes the assemblers, compilers, linkage editors, interpretive computer simulators, and any environmental simulators ESRO has developed. Each package is verified to ensure that it correctly operates in the STIL environment. Particular emphasis will be placed on assuring that STIL configuration control procedures will function with the ESRO software.

### 3.3.4 EGSE GROUND CHECKOUT INSTALLATION

The EGSE computer software is installed into the STIL data base. The software is verified using the support software discussed in Section 3.3.3. This activity ensures the ability of the STIL to verify future EGSE software activities and also trains support personnel to assume the operational phase when the EGSE is delivered to the CIS.

### 3.3.5 CDMS SUPPORT SOFTWARE INSTALLATION

The CDMS support software installation is much like that of the EGSE described in paragraph 3.3.3. The main difference is that there are many more interfaces to test and verify due to the more sophisticated user interface of the STIL realtime mode. Following the full verification of this software, the STIL is fully operational.



### 3.3.6 EM SOFTWARE INSTALLATION

The EM software installation on STIL is completed prior to the EM arrival at the CIS. This software includes the CDMS operating system as well as the application programs to execute in the experiment, subsystem, and EGSE computers for hardware integration. As previously discussed, this software is made available by ESRO. The software is installed and verified on the STIL to ensure compatibility of test verification procedures and to provide training. The EM software will then be validated at the CIS prior to being transported to KSC.

### 3.4 OPERATIONAL SOFTWARE DEVELOPMENT LIFE CYCLE FLOW

During the Spacelab operational period, the majority of the Spacelab software will be baselined and utilized from flight to flight with little or no modification. Software required to support the individual experiments and unique payloads will be developed and/or modified throughout the Spacelab operational life.

The changing requirements for Experiment Flight Application software will in some cases result in minor modification or additions to the baselined software sets of the Subsystem CDMS, STIL, EGSE, POC and PPF. Figure 3-4 illustrates the interrelationships between these software systems and the facilities utilized during the definition, development and operation phases of the Spacelab operational software development flow.

#### 3.4.1 DEFINITION PHASE

The PI, during his definition phase, defines the services required to operationally support his experiment. As the requirements are defined, functional allocations are made to the proper software systems. If the baseline system cannot support the requirement, change activity is initiated. This logical flow is represented by the dotted lines in Figure 3-4.

The figure represents the situation where several independent definition phases will be in progress at the same time as illustrated by Define Experiment No. 1 and Define Experiment No. N blocks. Analysis of the Shuttle Mission Model shows that there may be in excess of 50 definition phases in process at one time. Conceptually, each will progress independently into the development phase.

#### 3.4.2 DEVELOPMENT PHASE

The normal development phase for each independent application proceeds in the development flow identified in Figure 3-4. Following the design, code and test, and verification, the software is baselined at the package level waiting for final flight assignment and CIS hardware integration.

Just prior to CIS hardware integration, the selected packages are combined into their proper sets. These sets are then verified into STIL to ensure interface and full compatibility prior to release to the CIS.



Set validation occurs at the CIs during the experiment hardware integration and hardware/software integration testing. The hardware and software are now operationally ready to support the mission.

### 3.4.3 OPERATIONS PHASE

The Operations Phase includes the Spacelab/Payload Integration, the Shuttle/Spacelab Integration, the flight, and Post Mission activities.

Spacelab/Payload Integration is the assembly of the Spacelab sections and integration of the payload with the sections. The final integrated systems tests are performed using the CDMS Ground Checkout Set and the EGSE Ground Checkout Set. Interface checks with the experiments are performed as required by the Experiment Flight Applications in the CDMS Flight Set.

Following the Spacelab/Payload Integration, the Spacelab is installed in the Shuttle. The CDMS Flight Set is used to perform interface checks during launch readiness verification. These tests are designed to detect damage that may have occurred during the move and installation.

During flight operations, the Subsystem Flight Set monitors the Spacelab while the Experiment Flight Set monitors and controls the experiments. Both may communicate with the Payload Operations Center packages via the orbiter.

After landing, the CDMS Flight Sets and the EGSE Ground Checkout Set are required for the post mission data dump and refurbishment at the Launch Site. Scientific Data Reduction packages operating at the Preprocessing Facility format the data for dissemination to the PI.

During the operational phase, NASA software will be required to provide realtime support to isolate and correct reported anomalies and/or develop workaround procedures. If this realtime activity results in detecting a condition which must be corrected, it will be scheduled for correction in a later baseline.

## EXPERIMENT FLIGHT APPLICATIONS SOFTWARE DEVELOPMENT PLAN 4

The NASA intent is to provide the PI with complete flexibility in developing the software required to operationally support his experiment. NASA will not be involved in any software required in support of experiment hardware development prior to the experiment being integrated at the CIS or in supporting post flight data analysis requirements.

NASA will be responsible for the mission software of all experiments integrated at the CIS. This concept is represented in Figure 2-1 along with the basic five development options provided to the PI. The options provided to the PI for Experiment Flight Application package development are:

- Option 1 - PI develops the package on the NASA STIL
- Option 2 - NASA/STIL team develops the package
- Option 3 - PI develops the package on a copy of the STIL
- Option 4 - PI develops the package on his STIL compatible computer and NASA provides a CDMS Simulator
- Option 5 - PI develops the package on his own computer and NASA supplies a CDMS Simulator

Options 3-5 are considered as PI software development offsite and will be utilized for development of pre-CIS integration, experiment definition, and data analysis software developments.

Due to the interrelationships between the PI options and the NASA responsibilities, a preliminary structure for control and responsibility must be established. The remainder of this section of the plan will address recommended controls and responsibilities for Experiment Flight Applications software.

### 4.1 EXPERIMENT APPLICATION SOFTWARE CONTROL

Past NASA experience indicates that it is desirable to have a Software Review Board (SRB) to interface between the formal NASA Change Control Boards (CCB), the user, and software developers. The SRB responsibilities are to coordinate all activities of Spacelab software to ensure compatibility in performance and schedule. The members of the board are both technical and managerial which provides quick resolutions and decision making. The SRB acts on the direction of the CCB. Only major problems must be forwarded to the CCB, such as major schedule and cost problems. Figure 4-1 represents the SRB as the hub of all Spacelab software activities. Representation data and control flow is depicted on the chart.

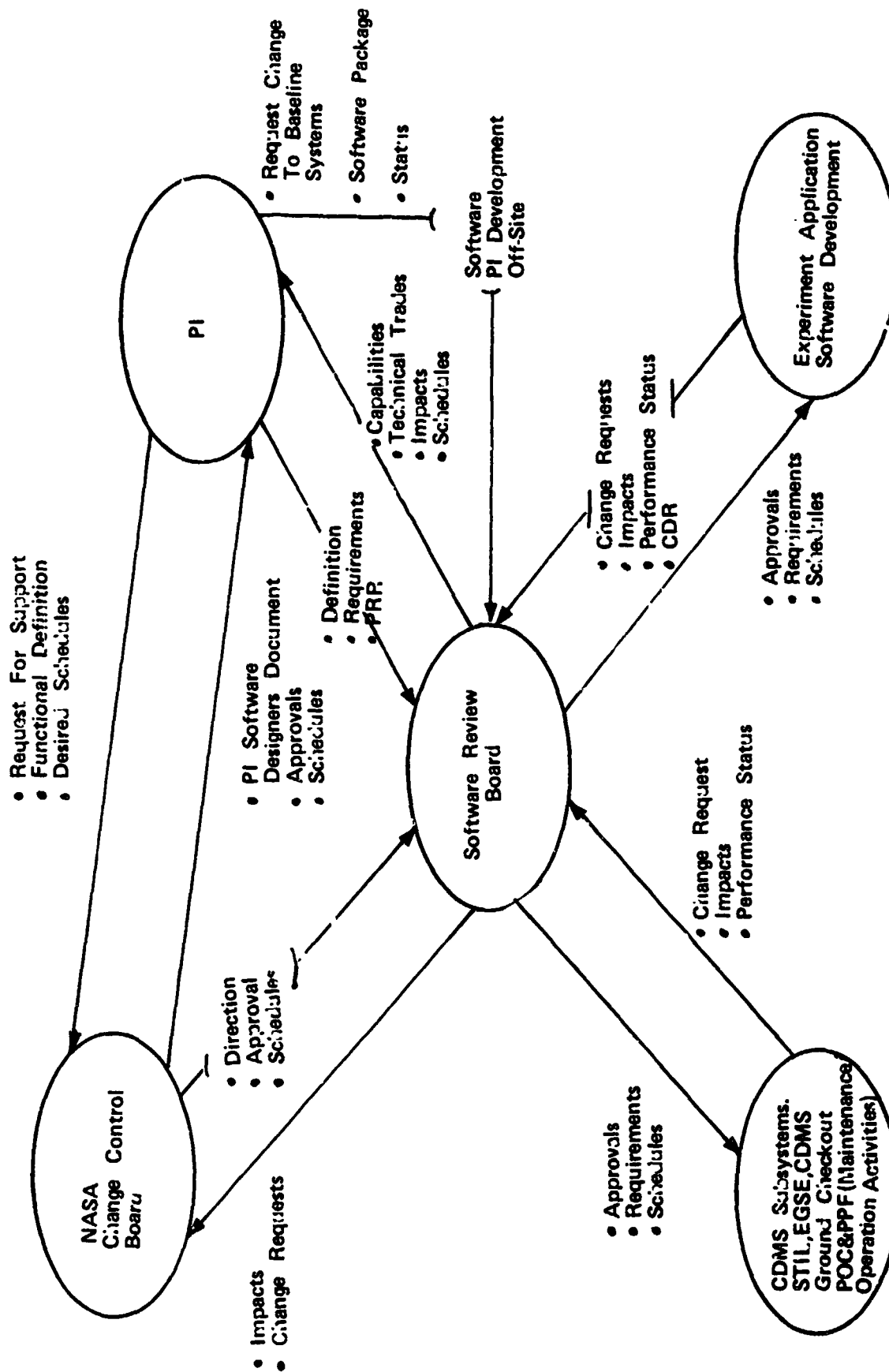


Figure 4-1 Basic Experiment Application Development Cycle Control and Data Flow

The PI will provide a PI's Software Designer's Document (SDD) at the time his experiment is selected to be flown. The SDD will define the complete software development environment including the services NASA will provide.

The PI, regardless of option selected, will be responsible for defining the operational software requirements and services he needs from NASA. Following approval of the CCB, the PI's main interface will be with the SRB on a technical level.

#### 4.2 EXPERIMENT APPLICATION SOFTWARE RESPONSIBILITIES AND INTEGRATIONS

The responsibility split between the PI and NASA/STIL team is defined in Table 4.1 and varies depending upon the option selected by the PI and CCB.

It should be noted, when the PI develops software on the STIL, he has the option of entering the configuration control at the module or package level. Should the PI select to develop software at his site, he must enter the configuration control cycle at the package level for NASA/STIL integration and verification. The NASA/STIL team will always become responsible for the configuration control and verification once the software element is placed into the library.

Table 4.1. Major Development Responsibilities for Experiment Flight Application Packages

	OPTION 1 - PI DEVELOPS ON NASA STIL	OPTION 2 - NASA TEAM DEVELOPS EXPERIMENT SOFTWARE	OPTIONS 3-5 - PI DEVELOPS ON OFF-SITE FACILITIES
Experiment Definition of Requirements and Performance	PI	PI	PI
Application Package Design			
Module Design	PI	STIL	PI
Module Code and Test	PI	STIL	PI
Module Verification	STIL/PI	STIL	PI
Module Library-Configuration Control	STIL	STIL	
Application Package Integration/Verification	STIL/PI	STIL	PI
Application Package Library Configuration Control	STIL	STIL	STIL
Flight Application Set Integration/Verification	STIL	STIL	STIL
Flight Set Library Configuration Control	STIL	STIL	STIL
Hardware/Software Validation	STIL	STIL	STIL

NASA will establish, document, and impose upon all Spacelab software developers a set of effective development standards. It is extremely important that these standards be developed very early to ensure compatibility and commonality. Spacelab software will be developed by several different groups in Europe as well as in the U.S.; therefore, it is imperative that the same standards be applied across all development activities to ensure that the resulting software is integratable and usable during the Spacelab operational phases. Recommended software development standards and techniques are documented in the Spacelab Software Development and Integration Concepts Study Report, Volume 1, IBM No. 73W-00326, October 31, 1973. A summary of these follow.

#### 5.1 SOFTWARE DEFINITION AND DESIGN PHASE

The standards, conventions, and practices established for software definition and design phase are the most critical. This is true because they establish the modularity that make the final product easy to code, test, verify and integrate. The following are recommended standards:

- o Insist on performance related specifications
- o Utilize composite design techniques
- o Design for HOL use
- o Prove design with development models
- o Use standard documentation formats
- o Use verified modules without modification.

#### 5.2 SOFTWARE IMPLEMENTATION PHASE

Software implementation is the coding and testing of the software. This activity will be done by a number of programmers at various skill levels. Once the software is coded and tested, it will be maintained by still different programmers. Therefore, it is important for low life-cycle cost that standard procedures and techniques be applied. The following are recommended:

- o Develop software top-down
- o Use HOL where possible
- o Make structured coding techniques mandatory
- o Make full use of program libraries
- o Perform testing using high fidelity simulators



- o Maintain documentation in listings
- o Standardize use of labels
- o Enforce coding restrictions.

### 5.3 VERIFICATION/VALIDATION PHASE

Software verification/validation costs have been high on past space programs. Within time and cost restrictions, new and improved methods of software verification must be developed to meet Spacelab schedule and cost objectives. The following are recommended baselines:

- o Perform verification top-down
- o Use cause and effect analysis concepts
- o Use automated tools to enforce standards
- o Develop verification data analysis programs.

### 5.4 DOCUMENTATION STANDARDS

Many forms of documentation standards can and will be imposed on the software developer. These standards can become significant cost drivers. Spacelab must generate documentation that is required and not what is desired. Every effort must be made to increase effectivity and reduce cost. The following should be considered:

- o Minimize and centralize software documentation
- o Structure documentation
- o Include documentation in the online data base with same configuration controls as software.

Software configuration management is much like hardware configuration management. The goal of configuration management is the formal control of a product from design to delivery to ensure the delivered end item meets requirements. For hardware, this control begins during design and continues through the development and buildup of parts, subassemblies, assemblies, subsystem and systems until a deliverable end item is produced. Configuration control continues during the production phase to track and control design changes.

An analogous cycle exists for software. Configuration control is established at the module level during the design phase and continues as the modules proceed through code and test, verification and validation. Configuration control continues through the buildup of modules into packages and packages into sets. The sets are then deliverable end items of the software development process. Following delivery, the maintenance of the delivered software set is analogous to the production phase for hardware. Configuration control is required to track and control design changes to modules to ensure that subsequent delivered software sets function in a reliable manner.

The NASA Spacelab Program Office will employ a configuration management system for the formal control of Spacelab software development, integration, and maintenance activities through the use of three procedural concepts: configuration control, configuration identification, and configuration accounting.

#### 6.1 CONFIGURATION CONTROL

The configuration control for the Spacelab program is the systematic control of the software development after establishment of a formal baseline.

Configuration control ensures that no changes are made to an approved baseline without a formal review and assessment of the proposed change by an appropriate Configuration Control Board (CCB). This assures that the Spacelab software modules remain unchanged until either errors in design are discovered or until a change is required which will modify the modules operating characteristics. In either event baseline requirements, design, and baseline end items are the norm against which configuration is controlled. Control is effected by tracking of a software identification number throughout a change cycle with evaluation and approval by the CCB.

##### 6.1.1 SPACELAB SOFTWARE CONFIGURATION CONTROL BOARD (CCB)

A CCB will be established to handle and process all software activity. Represented on the board will be technical and/or management representatives from the various areas effected by a proposed action. The board will be responsible for coordinating and scheduling all Spacelab software activities.

### 6.1.2 SPACELAB SOFTWARE PROCESSING CONTROL FLOW

The control of software is affected by formal review of major milestones in the software development process. These reviews enhance the total software development and provide the software development management visibility. The following formal reviews of software development is recommended:

- o Preliminary Requirements Review

A review held by the CCB to verify that initial requirements are compatible with Spacelab capabilities. The PRR will ensure sufficient resources are available to proceed with the design.

- o Critical Design Review

This review is conducted to verify that software has been designed in accordance with the requirements initially evaluated and agreed upon in the PRR and that software standards are being followed.

- o Customer Acceptance Readiness Review

This review establishes that all program requirements have been developed in accordance with design specifications and programming standards. It is the culmination of the verification process and the software is baselined.

### 6.2 CONFIGURATION IDENTIFICATION

The various elements of software must be marked and identified in a positive manner with a configuration identification number that is analogous to a hardware part and serial number. The configuration identification number is applied to each software module, which is the lowest configured element of Spacelab software. The configuration identification number is unique for each module in the STIL data base and when a modification is made to the data base, the configuration identification number is automatically revised. Thus, positive identification is maintained at all times on modules in the STIL data base.

Elements of software which must be identified and tracked are symbolic source code, compiled machine code, program listings, and program specifications for each module. Additionally, packages and set are positively identified. Simulation models are identified with the module, package, or set as appropriate.

### 6.3 CONFIGURATION ACCOUNTING

Configuration accounting is the element of configuration management that provides the essential records and the reporting of precise configuration status. The primary objectives of configuration accounting are as follows:

- o To maintain current and accurate configuration baselines and end item data

- o To maintain correlation and interface data among the various hardware, software, and support elements of the system
- o To maintain current and accurate records of the status of changes completed and in process.

In order to maintain, store and correlate these items, MSFC must prepare and update change documentation and keep the data base of configured modules, packages and sets. The accounting and reporting function must be automated and terminal driven to maintain the status of the massive amount of celab software.

## SOFTWARE DEVELOPMENT SUPPORT TOOLS 7

The proper tools and test equipment are as important to software manufacturing as to hardware manufacturing. A state-of-the-art software development facility will be required to meet the Spacelab software development objectives. This facility will require a host computer with an attached CDMS and extensive support software. The following data is a summarization of the data contained in Section 5 of the Study on Spacelab Software Development and Integration Concepts Final Report, Contract NAS8-30538, IBM No. 74W-00224.

### 7.1 STIL SUPPORT SOFTWARE REQUIREMENTS

The following software manufacturing and test tools are required to be STIL resident:

- o Realtime Interactive Tools

Realtime interactive software development tools provide the environment to meet the design and software development productivity requirements of Spacelab experiment development timelines. The realtime tools provide a dedicated "hands-on" environment to the software developer. Interactive tools should include the ability to dump, trace, stop, and single step the CDMS computer on the HOL statement as well as on machine language instruction level. This control is provided to the developer at an interactive graphics terminal which provides the ability for him to quickly make decisions and enable maximum productivity.

- o On-Line Interactive User Aids

On-line interactive user aids provides the tools necessary for the developer to make full use of the Host Computer batch processing services from remote locations. The ability to remotely submit corrections and review the STIL data base provides maximum efficiency.

- o Environment Models

The environment models simulate the environment in which the CDMS and EGSE computer software must function. The models will simulate various portions of the environment as required to test a particular software module, package or set. The currently identified models are Spacelab subsystems, Experiments interface, Shuttle Orbiter interface, Payload Operations Center interface, and EGSE interface.

- o Development Models

Development models are written in an HOL and are mathematical representations of the software to be developed. The models are used for concept and requirement testing during the software definition and design phase.

- o CDMS and ECSF Interpretive Simulators

A bit-for-bit interpretive computer simulator is required to provide complete repeatability for detailed software verification and problem resolution. The simulator will be used to verify detail timing and hardware/software interaction.

- o Functional Simulator

The functional simulator simulates the execution of the Experiment Flight Application software in the language of the STIL HOST computer. When executed, the functional simulator performs the same functions as the software being simulated and allows testing of software concepts in near realtime.

- o Experiment Application High Order Language (HOL) Compiler

The basic requirements of high productivity, fast integration, and large change activity dictates that a HOL be used for experiment application software development. The HOL provided must execute on the HOST computer and be capable of code generation for the Experiment Computer and FOST computer. The compiler must have built in error detection and be compatible with the functional simulator.

- o Experiment Simulation Language Compiler

Because of the diversity of the experiments to be supported, a language must be provided which will allow rapid development of environment models. This will be a table driven language to rapidly develop environment models of various experiments.

- o CDMS Support Software

This is basic software for developing CDMS software. The support software packages are an assembler, a compiler, and a linkage editor.

- o EGSE Support Software

An assembler, a compiler, and a linkage editor will be required for the EGSE. If the EGSE computer is identical to the CDMS computer, only one support package will be required.

- o On-line Source Data Management

The source data for every version and level of each module must be maintained as a unique member name in a partitioned data set. Updates to these modules must be either from terminal online or background batch update. In either method of data management each source module must have a directory entry which will uniquely identify the module by date, time, version, and revision number of the last change made to the source record.

- o Automated Configuration Management System

An Automated Configuration Management System will be required which will provide the development data base and supportive software to support multiple software configurations

Many experiments will be re flown; however, some of the applications will be upgraded from flight to flight while others will remain stable. Availability of a module library is required to simplify the problem of assembling and integrating the software modules into the application package and Flight Set required for a particular mission. The obvious advantage of a program library lies in the fact that software modules requiring no change can be integrated in parallel with the assembly and test of those modules requiring change.

- o Automated Release System

The capability to automatically generate a release of the software system for a given payload is required. This release procedure must produce source listings, object code, tapes and required documentation as well as identify all changes to software modules and create a history of the software associated with each mission. The automatic release system will generate reports identifying the baseline as well as all listings, load modules, and other elements which make up the delivery package. This system will significantly reduce clerical tasks for each delivery.

- o Automatic STIL Scheduling System

An automatic scheduling system must be provided which will be capable of collecting all the individual milestone and STIL resources needed to complete the task. The system must utilize this data to produce a STIL utilization schedule and forecast scheduling conflicts.

- o Host Computer Operating System and Support Software

The Host computer will require a multi-tasking operating system with terminal management and realtime executive control programs. A full range of Host support software must be provided including HOL's and JCL.

## 7.2 STIL HARDWARE REQUIREMENTS

The STIL hardware requirements include the following:

- o Host computer complex and peripheral devices
- o Data base storage
- o Graphic terminals

- o Remote terminals
- o Strip charts
- o Plotters
- o CDMS hardware



**APPENDIX B**  
**SPACELAB EXPERIMENT SOFTWARE FLOW**

**23 MAY 1974**

SPACELAB EXPERIMENT SOFTWARE FLOW

23 MAY 1974

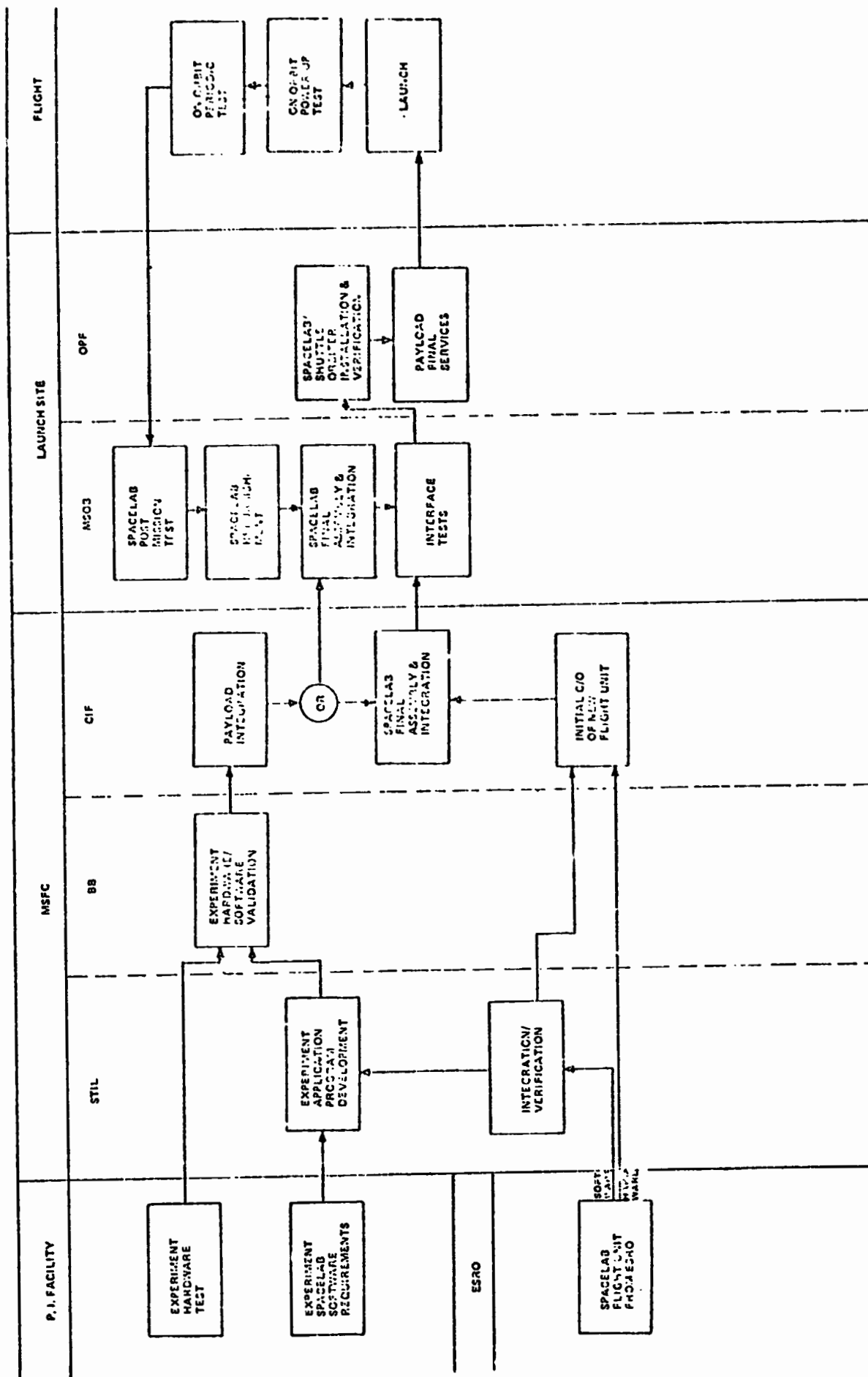
IBM

## PRESENTATION OVERVIEW

- EXPERIMENT SOFTWARE ANALYSIS OBSERVATIONS
- SPACELAB FLOW
- EXPERIMENT SOFTWARE INTEGRATION AND VALIDATION
- PAYLOAD TIMELINESS
- SOFTWARE INTEGRATION BOTTLENECKS
- CIS TIMELINES
- CONCLUSIONS

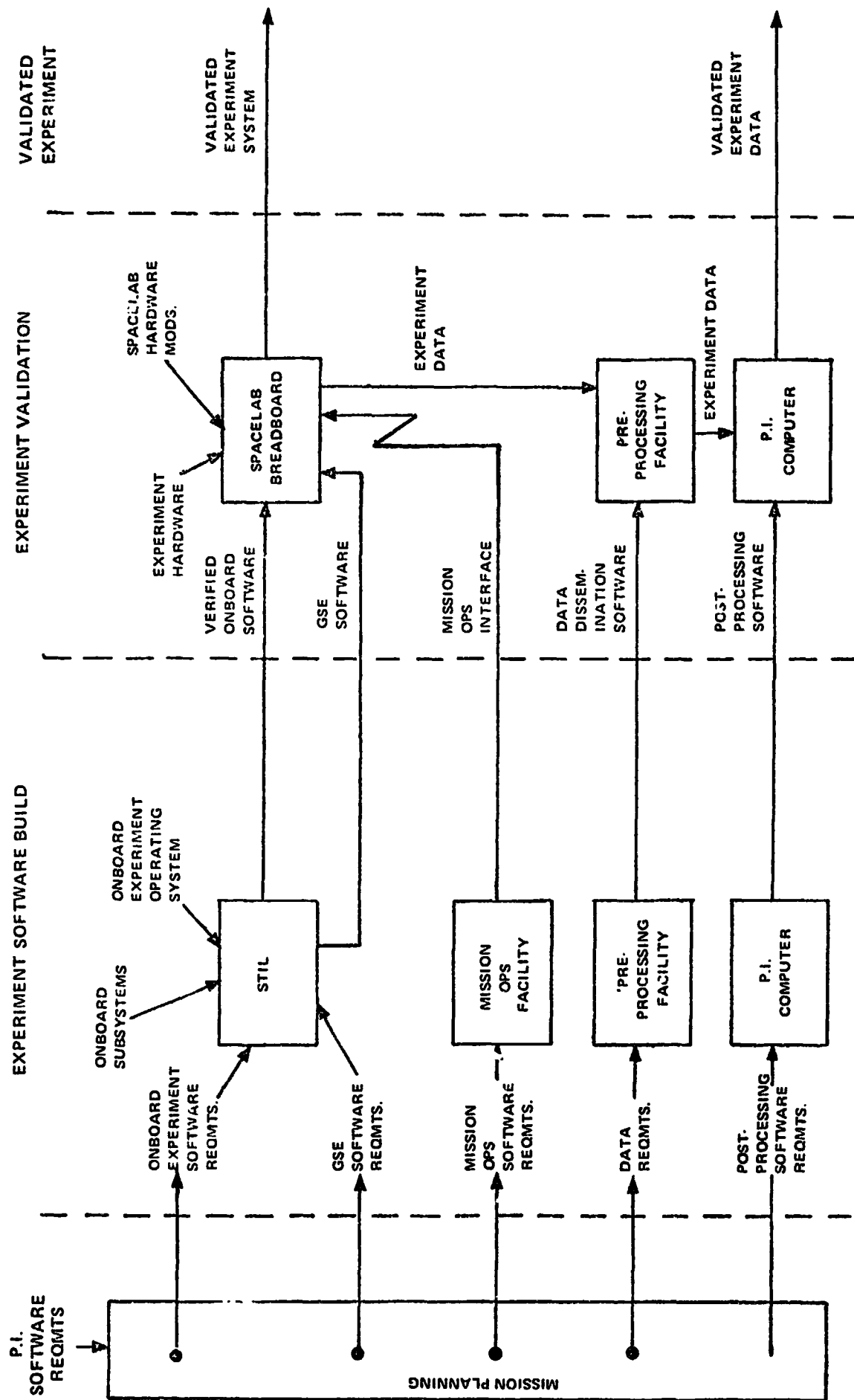
EXPERIMENT SOFTWARE ANALYSIS OBSERVATIONS

- INITIAL SPACELAB YEARS REQUIRE SIGNIFICANT SOFTWARE DEVELOPMENT (NO SLOW BUILD-UP)
- SPACELAB MISSIONS WILL BE FLOWN ABOUT EVERY TEN DAYS
- AVERAGE OF SEVEN NEW MISSIONS/YEAR AND 29 REFLIGHTS/YEAR
- EXPERIMENT CONTROL AND MONITOR SOFTWARE WILL BE REQUIRED FOR EACH NEW MISSION FLOWN
- REFURBISHED MISSION SOFTWARE WILL BE MODIFIED TO INCORPORATE EXPERIENCE GAINED ON PREVIOUS FLIGHTS



IBM

# Experiment Software Integration and Validation



## TIMELINE ANALYSIS ASSUMPTIONS

### BASED ON

- CURRENT TRAFFIC MODEL
- GROUND OPERATIONS PLAN
- TWO SHIFTS - FIVE DAYS PER WEEK
- SUPPORTED BY SIZING STUDIES

### FACTORS NOT INCLUDED

- GSE SOFTWARE MAINTENANCE
- EQUIPMENT DOWNTIME/MAINTENANCE
- HOLIDAYS
- INTERFERENCE
- HARDWARE/SOFTWARE DEVELOPMENT SCHEDULE SLIPPAGE
- TIME TO INSTALL SENSORS

### EXPERIMENT LOAD

- 3-18 SENSORS PER PAYLOAD - AVERAGE 8
- 80% OF SENSORS REQUIRE AN APPLICATION PROGRAM OR PROGRAMS
- SOME SENSORS MUST WORK IN UNISON
- SOME SENSORS MUST COMMUNICATE WITH A STANDARD POINTING CONTROL SOFTWARE SYSTEM

IBM

# STIL TIMELINE ANALYSIS PER PAYLOAD

## HOURS

	MIN	MAX	AVE	ALLOCATION	MIN	MAX	AVE	TOTAL
NEW PAYLOADS PER YEAR	2	10	7	40%	167	834	238	1668
NEW APPLICATIONS FOR REFURBISHMENT	9	41	29	10%	10	46	14	417
SOFTWARE VERIFICATION	11	46	36	30%	27	114	34	1251
SUBSYSTEM SOFTWARE MAINTENANCE	11	46	36	10%	9	38	12	417
SUPPORT SOFTWARE	11	46	36	5%	5	19	6	208
LIBRARY MAINTENANCE	11	46	36	5%	5	19	6	208
TOTAL TIME PER PAYLOAD	11	46	36	100%	90	379	116	4170

IBM



# PRELIMINARY

## SPACE SOFTWARE FACILITY REQUIREMENTS

MAJOR PROGRAMS	DEVELOPMENT FACILITY TIME (HRS)	VERIFICATION FACILITY TIME (HRS)	VALIDATION FACILITY TIME (HRS)	TOTALS (HRS)
SATURN FLIGHT (207A)	140	330	30	500
SKYLAB PREFLIGHT (16K)	455	200	100	755
SKYLAB FLIGHT (16K)	1,262	1,375	360	2,997
SHUTTLE ALT	1,200	400	-----	1,600
SPACELAB MISSION	252	34	12	298

IBM

NEW EXPERIMENT APPLICATION DEVELOPMENT  
TIME PER SENSOR

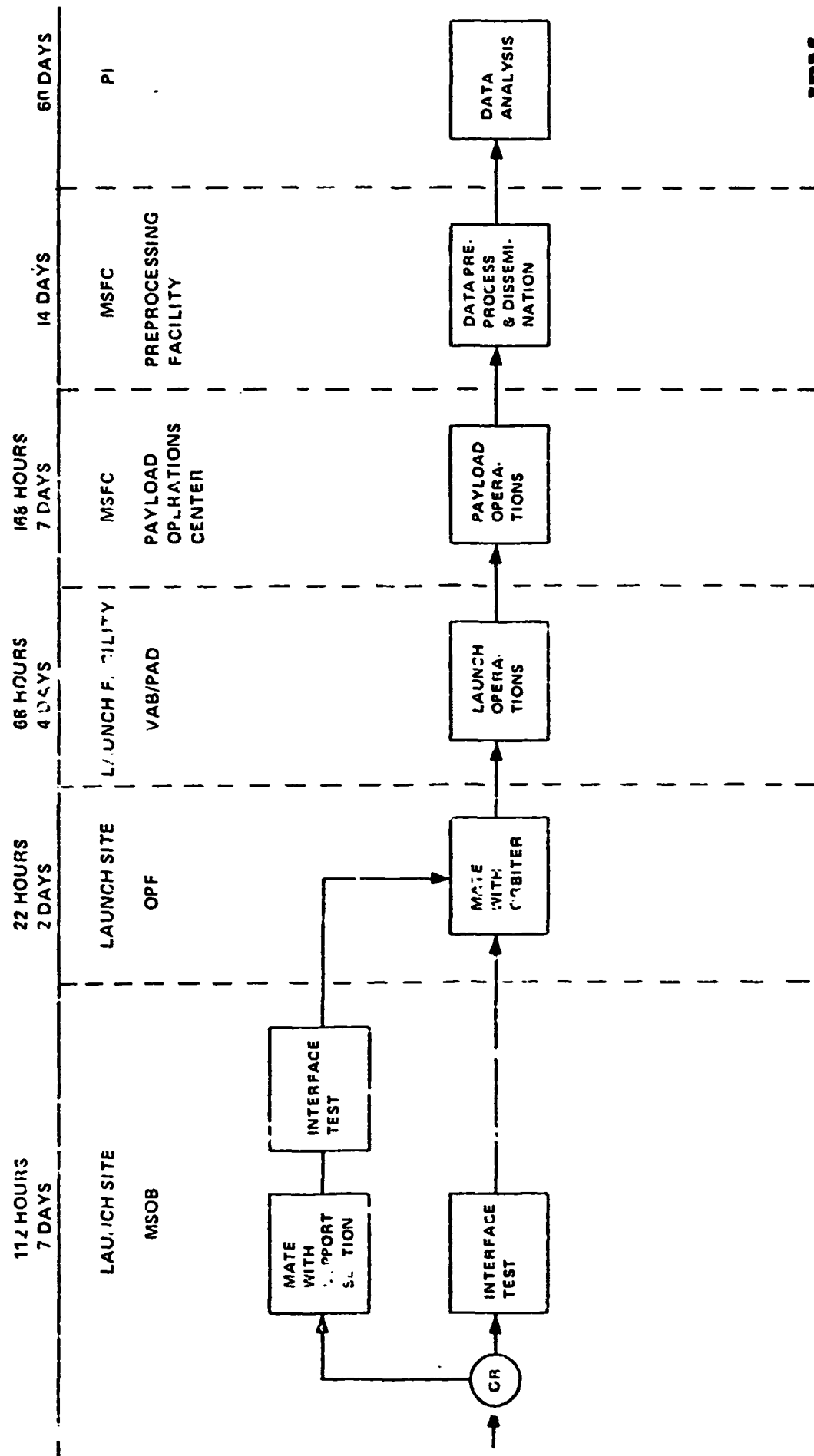
HOURS

		MIN	MAX	AVE
NEW PAYLOADS	WORSTCASE	(9)	35	21
	BESTCASE	46	(278)	104
	AVERAGE	18	79	(29)
REFURBS	WORSTCASE	(.5)	2.3	.8
	BESTCASE	2.3	(14)	4.6
	AVERAGE	1.3	5.2	(1.8)

\*TIME ON STIL CAN BE SPREAD OVER SEVERAL MONTHS

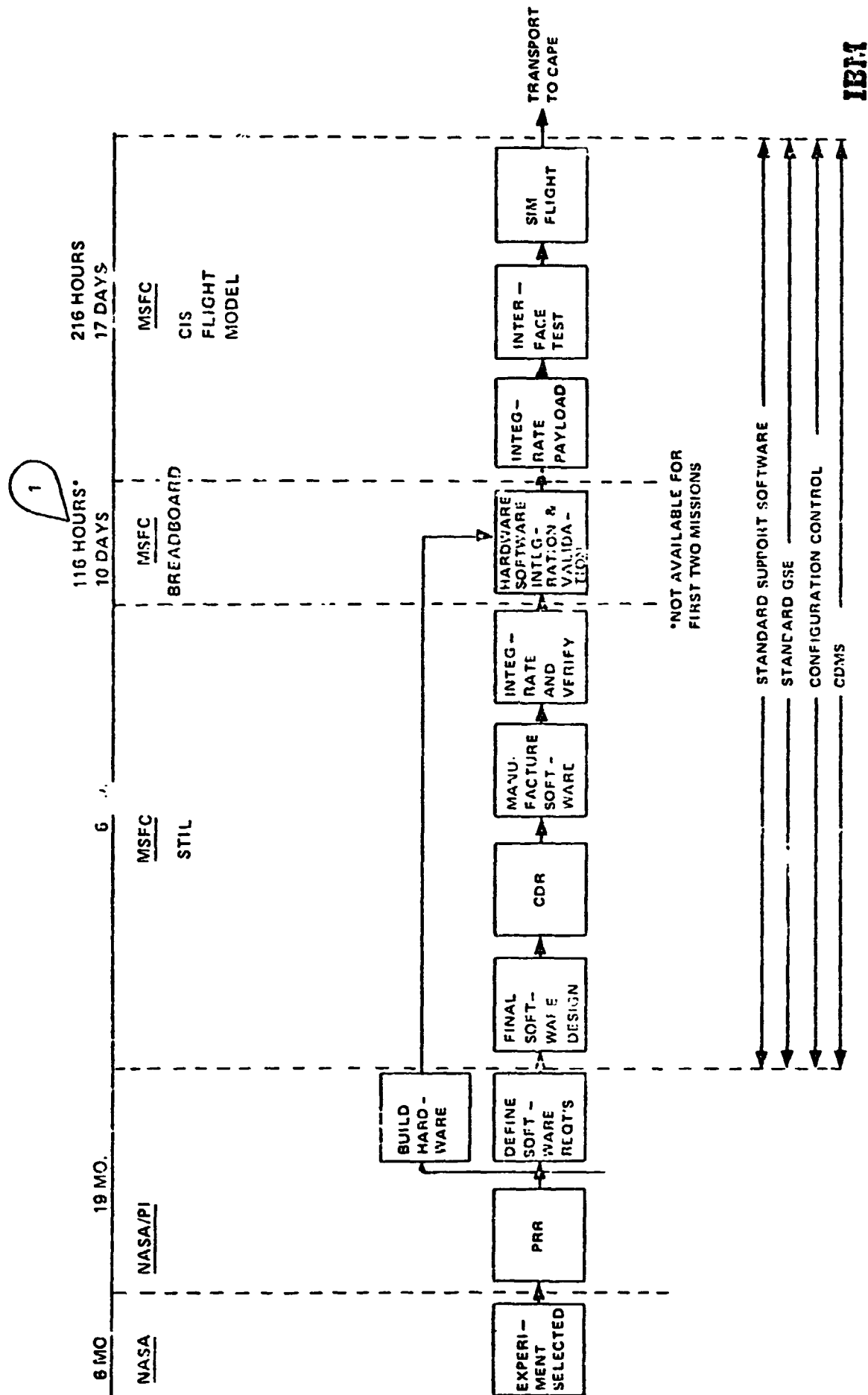
IBM

# Launch And Prelaunch Timeline

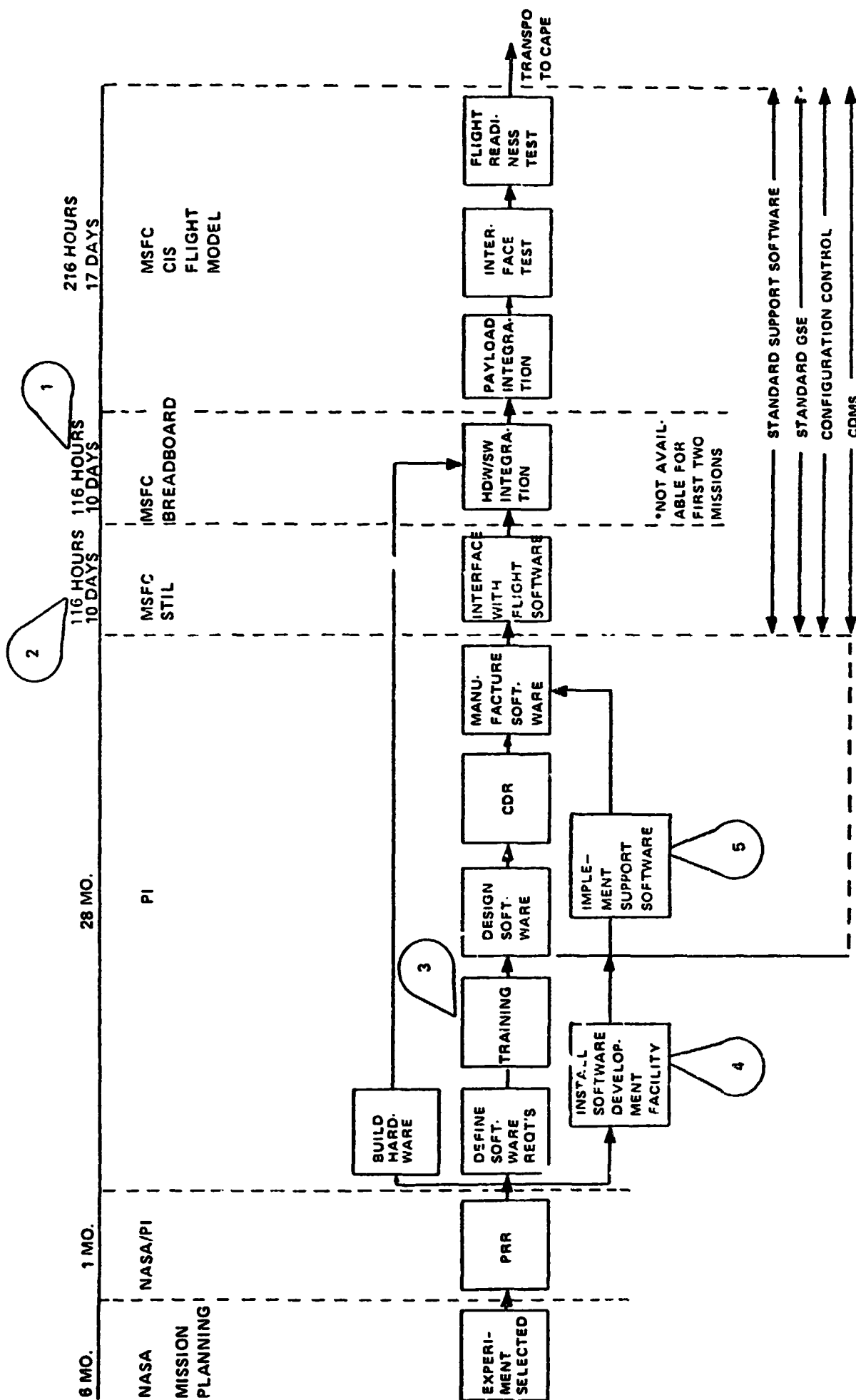


IBM

# MSFC Flight Software Development Timeline



# PI Flight Software Development Timeline At His Facility



IBM

OBJECTIVE OF EXPERIMENT FLIGHT SOFTWARE DEVELOPMENT/INTEGRATION

- MEET MISSION SCHEDULES
- ESTABLISH CENTRAL CONTROL OF SOFTWARE
- REDUCE CRITICAL FLIGHT SOFTWARE/HARDWARE INTEGRATION SCHEDULE
- ELIMINATE MULTIPLE SOFTWARE DEVELOPMENT INTEGRATION PROBLEMS
- ENSURE STANDARDIZATION
- MINIMAL TRAINING
- REDUCE NUMBER OF SOFTWARE DEVELOPMENT FACILITIES

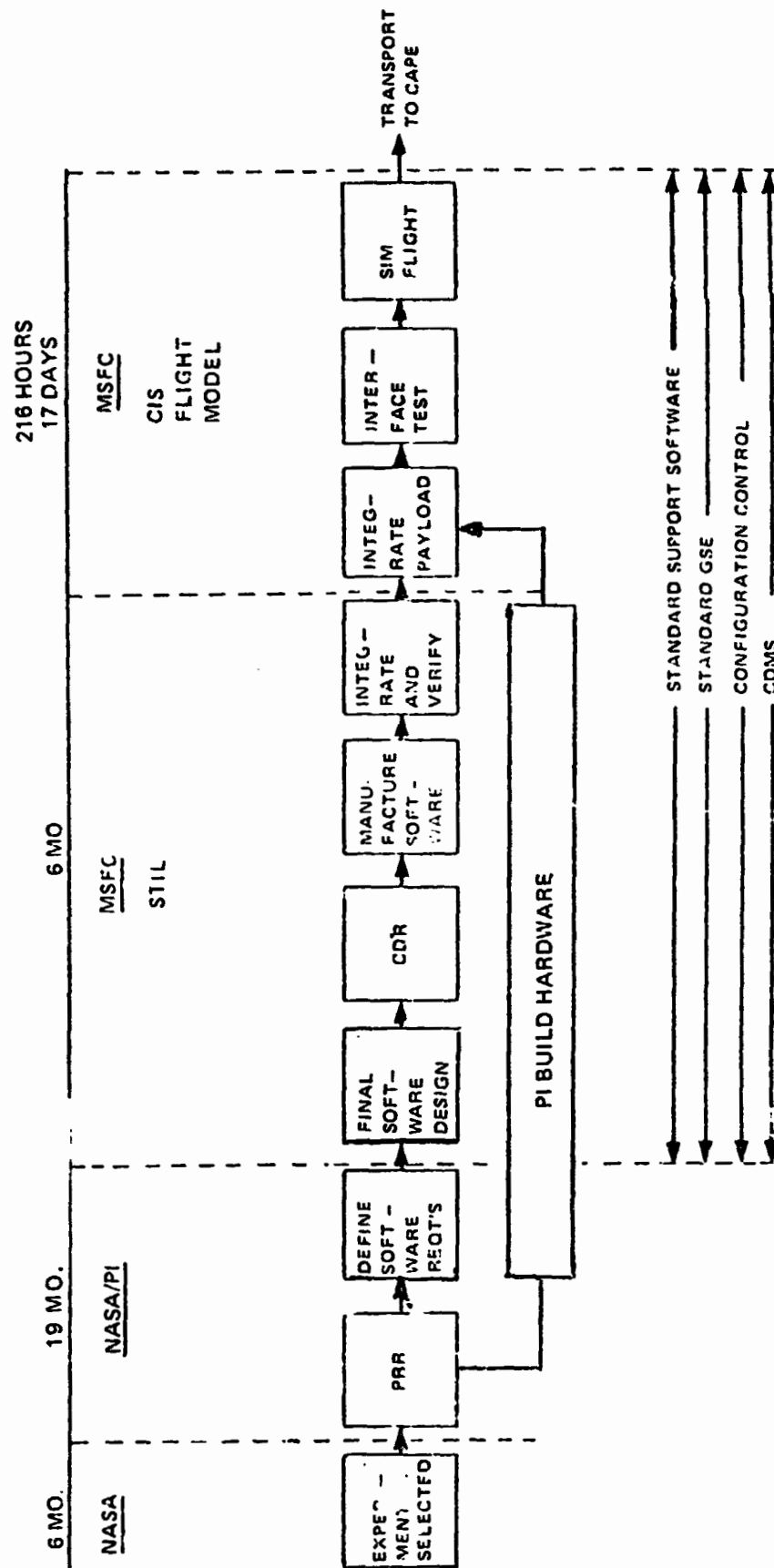
IBM

POSSIBLE APPROACHES

- USE EM TO DEVELOP EXPERIMENT PROTOTYPES AND ADVANCED CONCEPTS
- INTEGRATE EXPERIMENT HARDWARE/SOFTWARE ON FLIGHT EQUIPMENT
- MSFC DEVELOP EXPERIMENT APPLICATION SOFTWARE
- HIGH FIDELITY STILL SIMULATORS

**IBM**

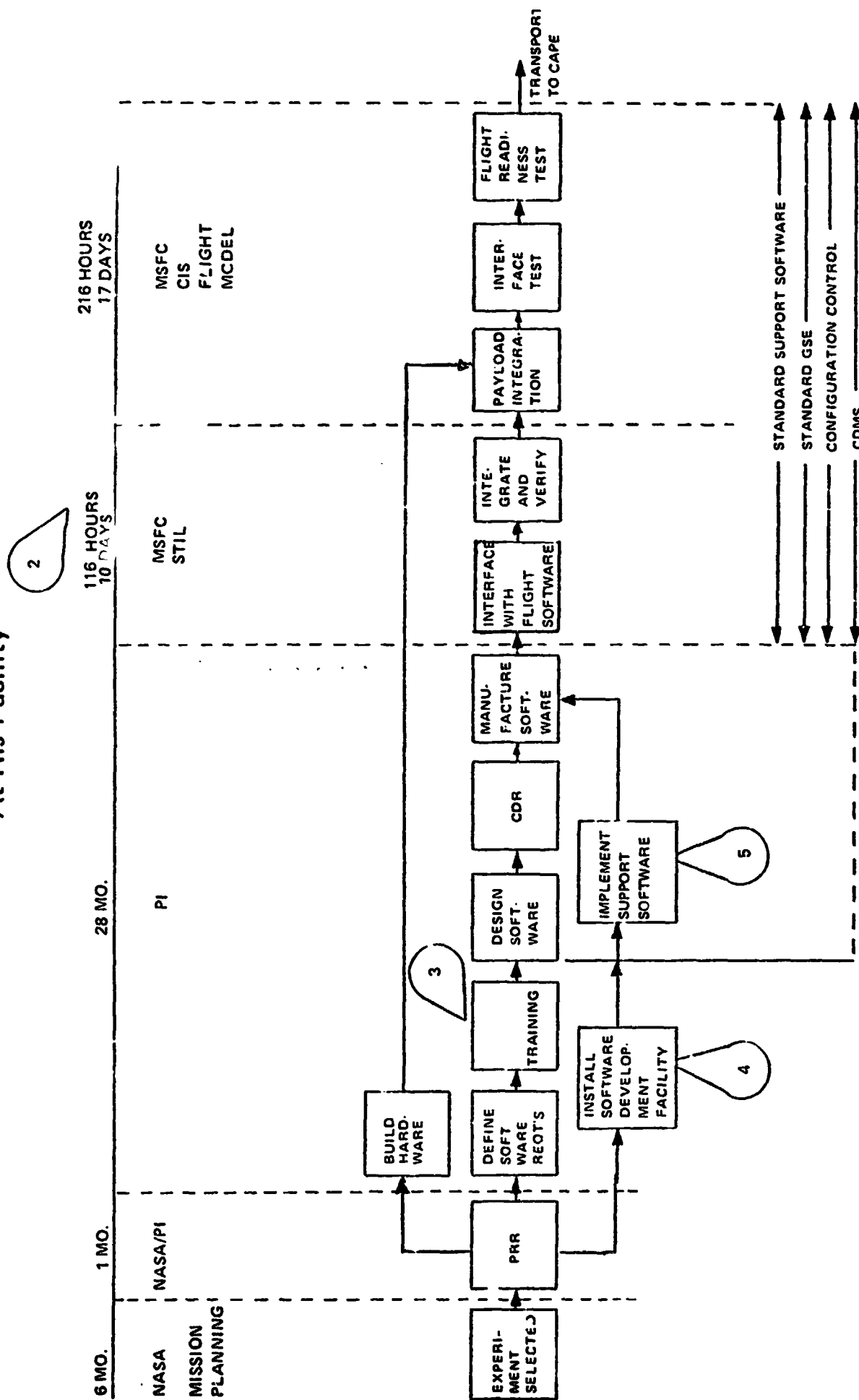
# MSFC Software Development Timeline



IBM



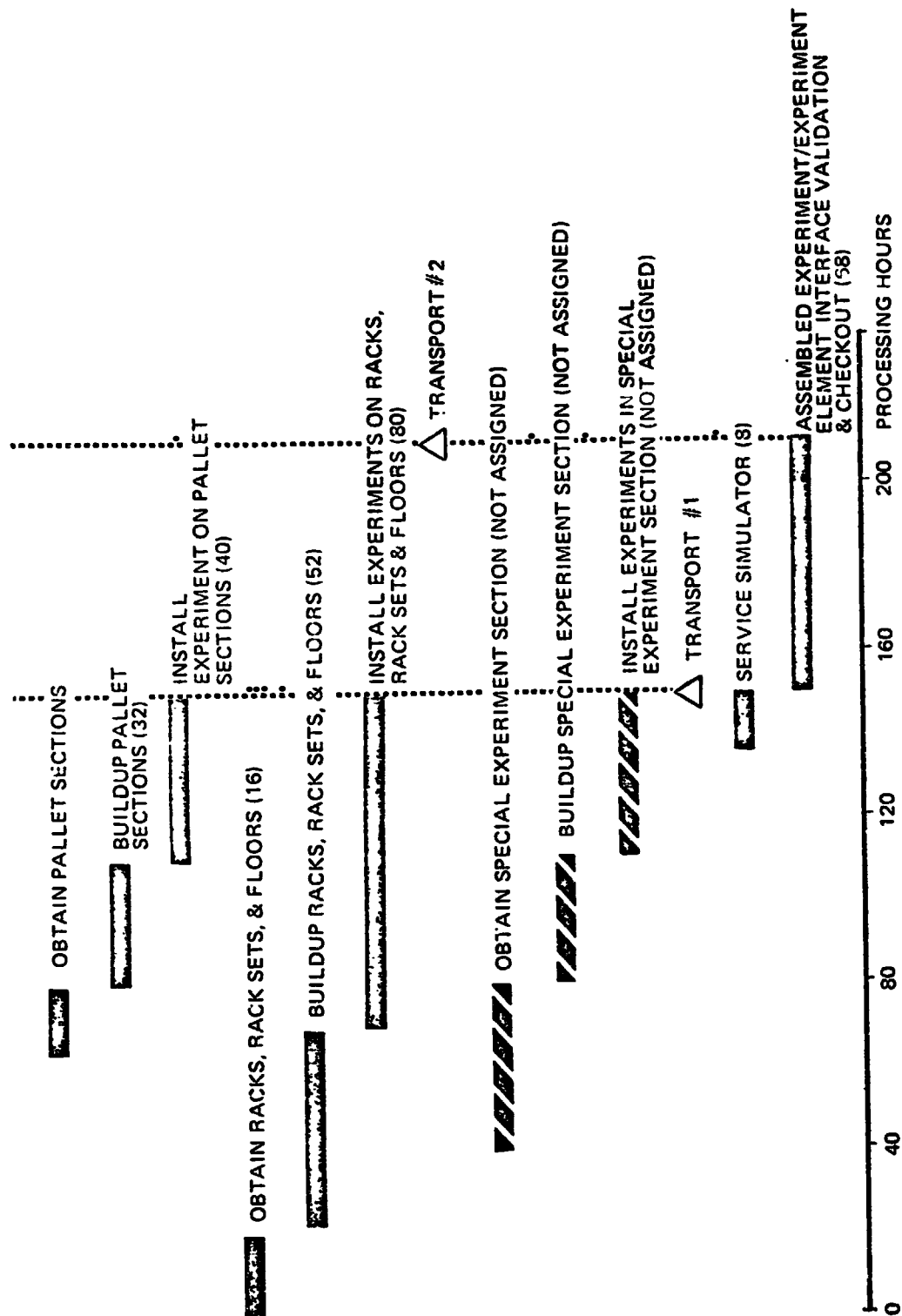
# PI Software Development Timeline At His Facility



CIS  
2 FLOW  
TURNAROUND CAPABILITY

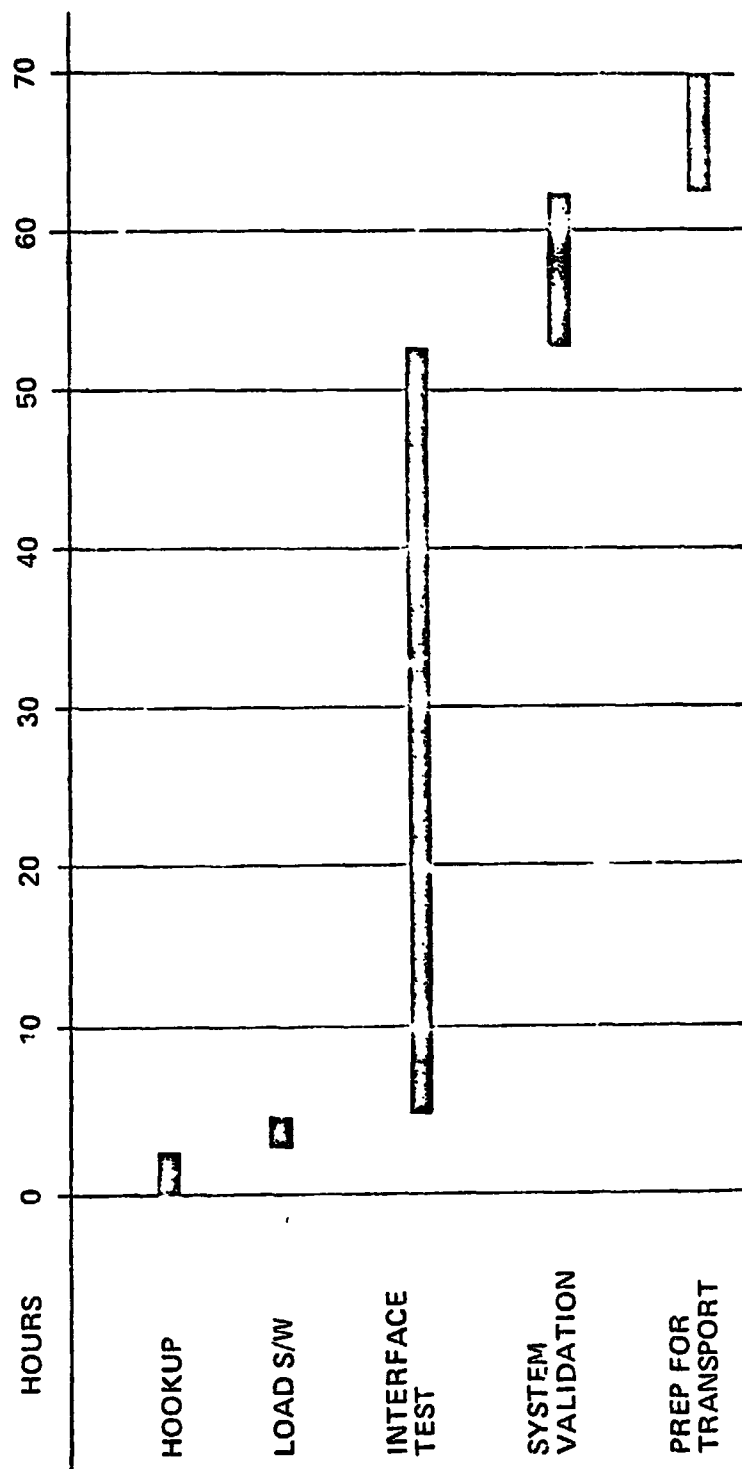
- 18 PAYLOADS PER FLOW PER YEAR
- 1 PAYLOAD EVERY 20 CALENDAR DAYS PER FLOW
- 216 HOURS PER PAYLOAD

# CIS TIMELINE



IBM

# EXPERIMENT/EXPERIMENT ELEMENT INTERFACE VALIDATION AND CHECKOUT



IBM

## CONCLUSIONS

- CHALLENGES EXIST IN THE TRAFFIC FLOW
- MORE IN-DEPTH ANALYSIS OF TIMELINES REQUIRED
- IN-DEPTH STUDY OF EXPERIMENT FLOW REQUIRED

**APPENDIX C**

**SPACELAB SOFTWARE TEST & INTEGRATION LABORATORY  
(STIL)**

**23 MAY 1974**

SPACELAB  
SOFTWARE TEST AND INTEGRATION LABORATORY  
(STIL)

23 MAY 1974

C-1

IBM

## PRESENTATION OUTLINE

- REQUIRED OPERATIONAL SOFTWARE
- REQUIRED SUPPORT SOFTWARE
- TYPICAL SOFTWARE DEVELOPMENT CYCLE
- BASELINE STIL CONFIGURATION
- SCHEDULE ASSUMPTIONS
- STIL DEVELOPMENT SCHEDULE

IBM



REQUIRED OPERATIONAL SPACELAB SOFTWARE

- SUBSYSTEM
- EXPERIMENT EXECUTIVE
- EXPERIMENT APPLICATIONS
- GROUND SUPPORT EQUIPMENT
- CIS ACCEPTANCE TEST SOFTWARE
- MISSION PLANNING
- MISSION OPERATIONS
- CREW TRAINING
- POST FLIGHT DATA PROCESSING

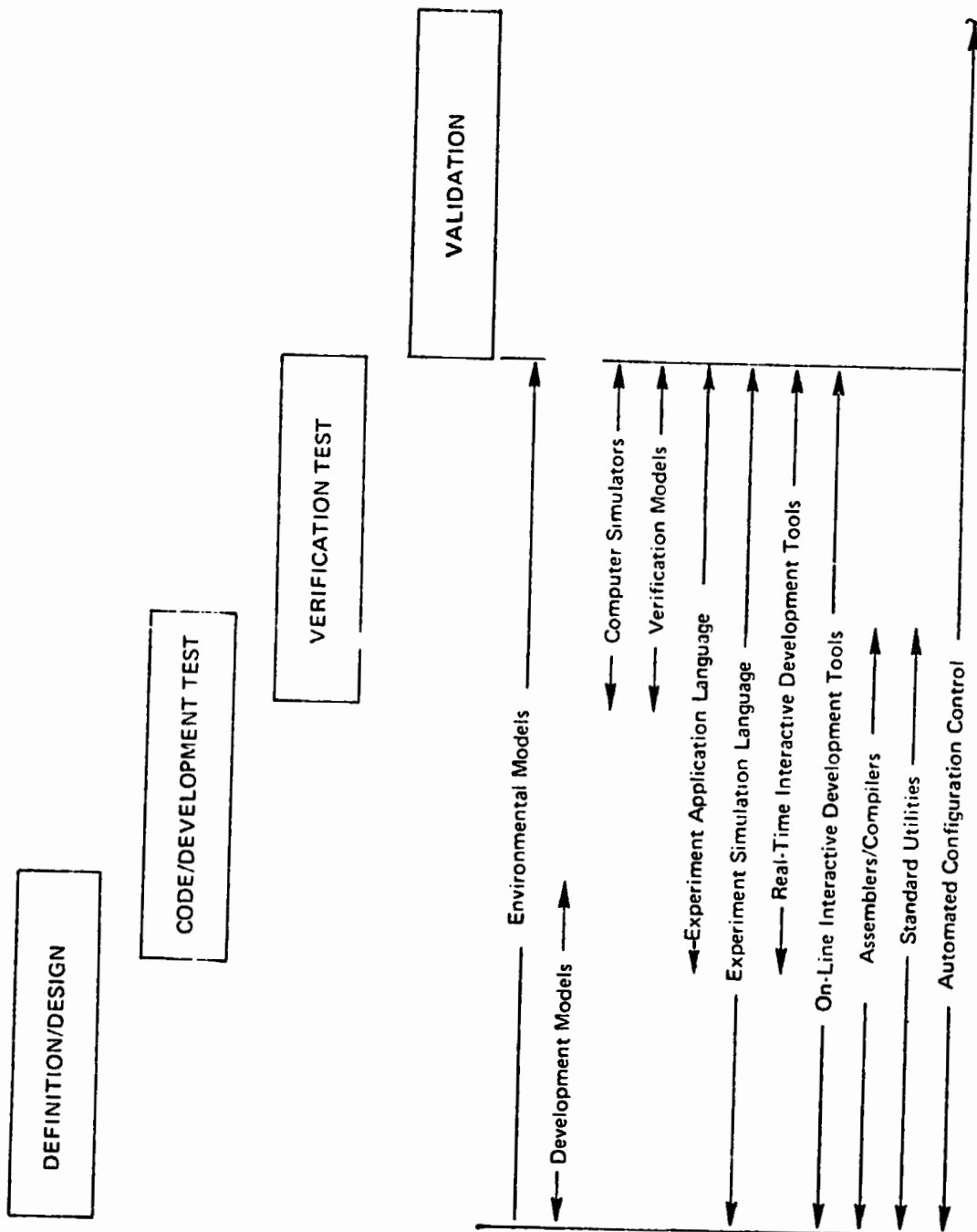
IBM

REQUIRED SUPPORT SOFTWARE AND DEVELOPMENT TOOLS

- ENVIRONMENTAL MODELS
- DEVELOPMENT MODELS
- VERIFICATION MODELS
- CDMS COMPUTER SIMULATORS
- EXPERIMENT APPLICATION LANGUAGE
- EXPERIMENT SIMULATION LANGUAGE
- REAL-TIME INTERACTIVE DEVELOPMENT TOOLS
- ON-LINE INTERACTIVE DEVELOPMENT TOOLS
- ASSEMBLERS/COMPILERS
- STANDARD UTILITIES
- COMPUTER SIMULATORS
- AUTOMATED SOFTWARE MANAGEMENT SYSTEM

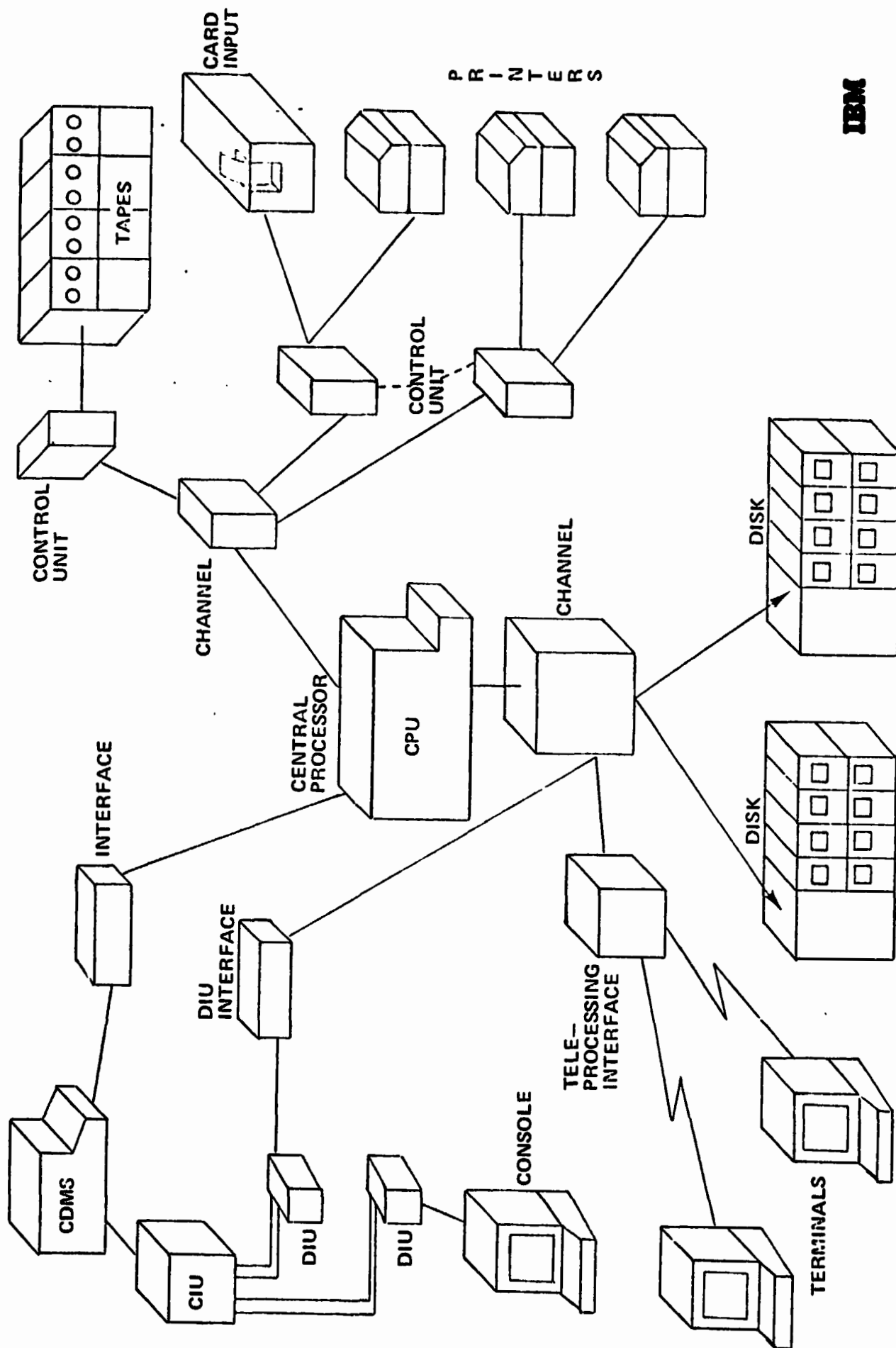
**IBM**

# TYPICAL SOFTWARE DEVELOPMENT CYCLE



IBM

# SOFTWARE TEST AND INTEGRATION LABORATORY (STIL)

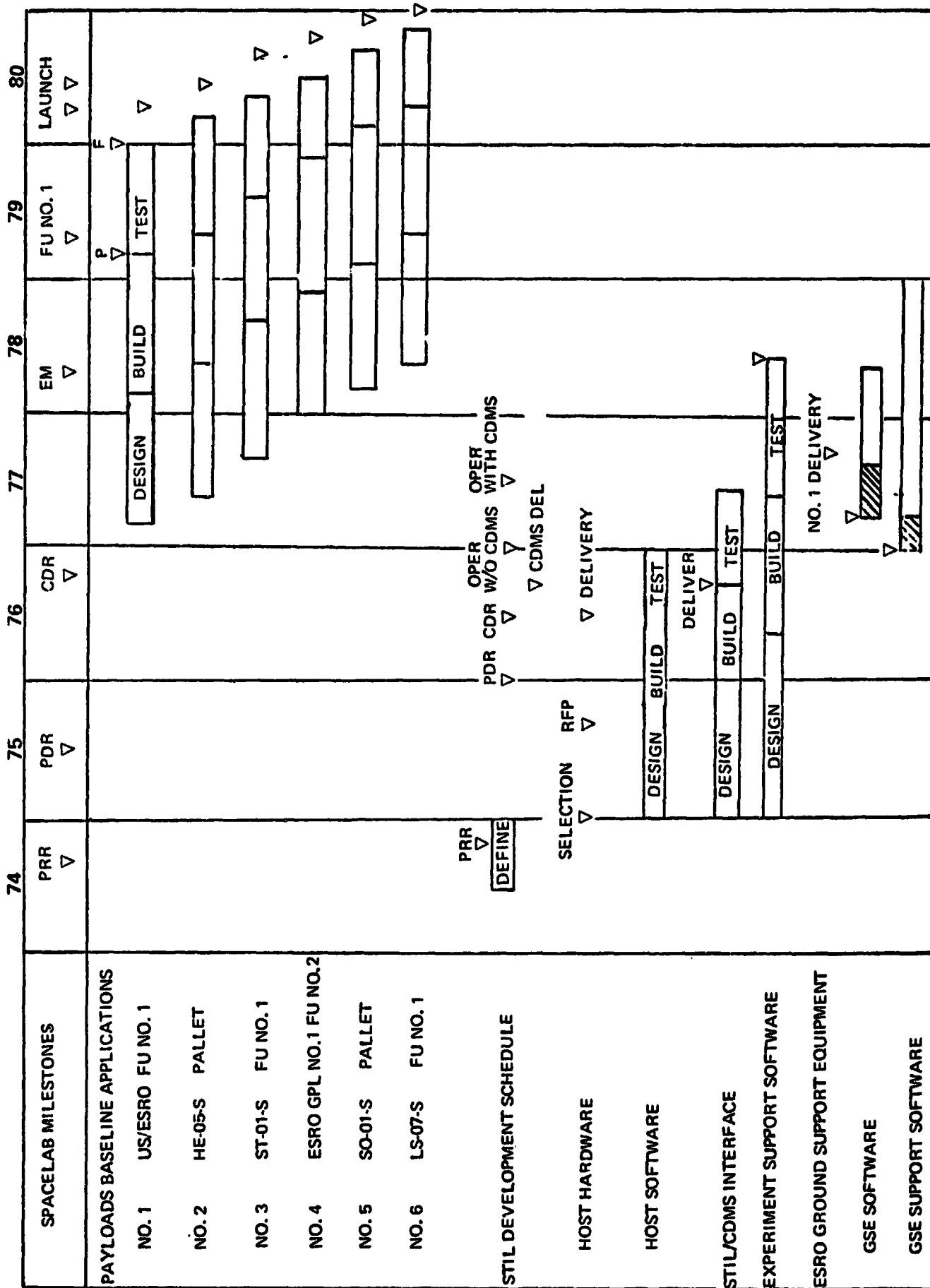


SCHEDULE ASSUMPTIONS

- ESRO WILL DEVELOP SUBSYSTEM SOFTWARE
- ESRO WILL DEVELOP EXPERIMENT EXECUTIVE
- ESRO WILL DEVELOP GSE SOFTWARE
- ESRO WILL SUPPLY SUPPORT SOFTWARE COMPATIBLE WITH STIL
- NASA WILL DEVELOP PAYLOAD APPLICATIONS
- NASA WILL DEVELOP EXPERIMENT APPLICATION SOFTWARE SUPPORT TOOLS
- NASA WILL DEVELOP ADDITIONAL SUBSYSTEM TEST PROGRAMS
- NASA WILL DEVELOP ADDITIONAL GSE SOFTWARE INCLUDING SIMULATORS
- NASA MUST PROVIDE CAPABILITY TO MAINTAIN ESRO-DELIVERED SOFTWARE

IBM

# STIL DEVELOPMENT SCHEDULE



5-31-74

**APPENDIX D**

**REFERENCE DATA**

1. LIST OF REFERENCES

1. NASA, Spacelab Ground Operations Plan, 68M00032, March 28, 1974
2. IBM, Program Management Plan for the Spacelab Software Development and Integration, IBM No. 74W-00105, April 4, 1974
3. IBM, Spacelab Software Development and Integration Concepts Study Report Volume I, IBM No. 73W-00326, October 31, 1973.
4. NASA-ESRO, Spacelab Programme Requirements Level I, March 5, 1974
5. NASA-ESRO, Spacelab System Requirements Level II, March 1, 1974
6. IBM, Spacelab Data Management Study, IBM, No. 73W-00314, October 15, 1973
7. IBM, Space Station Data Flow Amended for Spacelab User Interaction Study, IBM No. 74W-00044, February, 1974
8. M. S. Malkin, "Space Shuttle/The New Baseline," Astronautics & Aeronautics, January, 1974
9. IBM, Space Ultra-reliable Modular Computer (SUMC) Support Software Study, IBM No. 72W-00192, May 31, 1972
10. IBM, Concept Verification Test Subsystem Breadboard XDS-930 Operating System (BASIC) Detail Design Specification, IBM No. 71W-00393, January 1, 1972
11. IBM, Space Ultra-reliable Modular Computer (SUMC) Measurement and Control List Requirements Specification and Program Definition, IBM No. 74W-00036, January 15, 1974
12. IBM, Control and Display Language Description and Specification, IBM No. 73W-00369, December 21, 1973
13. IBM, SUMC Operating System Design Description, IBM No. 73W-00353, December 7, 1973
14. IBM, Saturn Software Systems Development Study, IBM No. 72W-00403, December 8, 1972
15. IBM, Checkout Programming Language Description and Specification, IBM No. 73W-00368, December 21, 1973
16. NASA, Space Shuttle Program - Configuration Management Requirements, Level II Program Definition and Requirements Volume IV, JSC-07700, March 6, 1974



17. NASA, Space Shuttle Program - Information Management Requirements, Level II Program Definition and Requirements Volume IV, JSC 077000, March 6, 1974
18. NASA, Space Shuttle Program - Shuttle Master Verification Plan, Computer Systems and Software Verification Plan Volume IX, JSC-07700-MVP-09, April 1973
19. NASA, Space Shuttle Program - Computer Systems and Software Requirements Level II Program Definition and Requirements Volume XVIII, JSC-07700, February, 1972
20. NASA, Space Shuttle Program - Computer Systems and Software Requirements Level II Program Definition and Requirements XVIII, Program Allocation of Computational Functions, JSC - 07700, April 1973
21. NASA, Space Shuttle Program - Computer Systems and Software Requirements Level II Program Definition and Requirements Volume XVIII, Software Management and Control, JSC-07700, April 1973
22. NASA, Space Shuttle Program - Computer Systems and Software Requirements Level II Program Definition and Requirements Volume XVIII, Software Development Standards, JSC-07700, April 1973
23. ESRO/CERS, Spacelab Request for Proposal A0/600 for the Design and Development Contract (Phase C/D), Appendix 1.1 System Requirements
24. MSFC, The October 1973 Space Shuttle Traffic Model, TMX-64751, Rev. 2, January 1974.
25. NASA/ESRO, Joint NASA/ESRO Spacelab Payload Computer and Display Requirements as Approved by JURE, 14/15 May 1974.
26. IBM, "Spacelab Sortie Payload Sizing Analysis," IBM, Contract No. NAS8-14000, IBM No. 74W-00059, DRL No. 1615, 27 February 1974.
27. ERNO Proposal for the Spacelab Design and Development Contract to ESRO/ESTEC, RFP A0/600, Volume I, Technical Proposal, April 16, 1974.

## 2. LIST OF ACRONYMS

ATE	Automatic Test Equipment
BITE	Built-In Test Equipment
CDF	Configuration Data File
CDR	Critical Design Review
CDMS	Command and Data Management System
CID	Computer Interface Device
CIS	Central Integration Site
CPU	Central Processing Unit
CRT	Cathode Ray Tube
EFA	Experiment Flight Applications
EGSE	Electrical Ground Support Equipment
EM	Engineering Model
ESRO	European Space Research Organization
ESTEC	European Space Research & Technology Center
GPSS	General Purpose System Simulator
ICS	Interpretive Computer Simulator
I/O	Input/Output
LRU	Lowest Replaceable Unit
MBPS	Million Bits per Second
MCE	Mission Control Executive
MIPS	Million Instructions per Second
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration

PI	Principal Investigator
POC	Payload Operations Center
PPC	Preprocessing Center
RAU	Remote Acquisition Unit
RF	Radio Frequency
STIL	Software Test and Integration Laboratory
STS	Space Transportation System
TCMD	Telecommand
TDRSS	Tracking and Data Relay Satellite Station